AUTOMATED AGENT ONTOLOGY CREATION FOR DISTRIBUTED DATABASES

THESIS

Austin A. Bartolo, First Lieutenant, USAF

AFIT/GCS/ENG/04-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/04-01

AUTOMATED AGENT ONTOLOGY CREATION FOR DISTRIBUTED DATABASES

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Austin A. Bartolo, BS

First Lieutenant, USAF

March 2004

AFIT/GCS/ENG/04-01

AUTOMATED AGENT ONTOLOGY CREATION FOR DISTRIBUTED DATABASES

Austin A. Bartolo, BS

First Lieutenant, USAF

Approved:

/SIGNED/	8 Mar 04
Dr. Gilbert L. Peterson (Chairman)	Date
/SIGNED/	8 Mar 04
Lt. Col Michael L. Talbert (Member)	Date
/SIGNED/	8 Mar 04
Maj Rusty O. Baldwin (Member)	Date

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my faculty advisor, Dr. Gilbert Peterson, for his guidance and support throughout the course of this thesis effort. The insight and experience was certainly appreciated. I would like to thank my committee members, Lt Col Talbert and Maj Baldwin for taking the time to provide great feedback. I would also like to thank my family for without their love and support this research would not have been possible. I would like to thank Maj Mayer and Mr. Speckman for their outstanding support during this endeavor.

I would also like to thank the AFRL Information Directorate, the Joint Battlespace Infosphere (JBI) team, Rome research site, Rome, NY for their outstanding support.

Finally, I would like to thank my fellow classmates. Without their help and support, I would have never made it through this program.

Austin A. Bartolo

Table of Contents

Page

List of Figures

Page

List of Tables

Page

AFIT/GCS/ENG/04-01

Abstract

In distributed database environments, the combination of resources from multiple sources requiring different interfaces is a universal problem. The current solution requires an expert to generate an ontology, or mapping, which contains all interconnections between the various fields in the databases. This research proposes the application of software agents in automating the ontology creation for distributed database environments with minimal communication. The automatic creation of a domain ontology alleviates the need for experts to manually map one database to other databases in the environment. Using several combined comparison methods, these agents communicate and negotiate similarities between information sources and retain these similarities for client agent queries without the manual mapping of different data sources achieving an average accuracy of 57% before leader negotiation and an average accuracy of 61% after leader negotiation. The best matching accuracy achieved in a single test is 79%.

This is directly applicable to the Department of Defense (DOD) that possesses many systems, which share information that enables the military to achieve their objectives. The DOD created an environment called the Joint Battlespace Infosphere (JBI) to solve this integration problem. This research improves upon the JBI's use of exact matching of field names for integrating the information within the environment. It simulates this type of interaction by demonstrating agents wrapped around different databases negotiating and generating an ontology. An agent-generated ontology is compared with an expert generated ontology and testing uses a set of queries run against the ontologies show that this technique can be useful in a distributed information environment.

AUTOMATED AGENT ONTOLOGY CREATION FOR DISTRIBUTED DATABASES

1. Introduction

The United States is a technology-based society. Computers, cell phones, televisions, and stereos pervade the country. The smaller the device and the more features it has, the more attractive the device seems to be to the consumer.

In the 1950s, computers filled large rooms and only large corporations could afford them. In the late 1980s and early 1990s, high-speed computers fit on a desk but were still expensive. For example, in 1992, computer memory sold for \$30.00 to \$45.00 per megabyte (MB) and hard drives sold at \$1 per MB. Today, technology is smaller, cheaper, and faster than ever and millions of Americans own computers and communicate via the internet.

The U.S. Department of Defense (DOD) has seen this same technological advancement as they harness this technology into a war fighting capability. A major problem is command and control's (C2) 'lack of interoperability'. In wartime, quick and efficient flow of timely information is a decisive factor in victory. Communication has been difficult in situations involving two or more services. Today when a military conflict arises the US military must be able to act together in a coordinated effort to accomplish military objectives. This is called joint operations.

Joint operations require a joint language. The Joint Battlespace Infosphere (JBI) is the conceptual framework the military uses to consolidate information resources. JBI provides a repository wherein a user can query and receive every piece of information needed from a distributed collection of information resources instead of querying each entity individually. While the JBI concept focuses on assisting the many systems of the military with interaction in a common environment [1], the real problem is the integration of these systems. Manual mapping database attributes is a solution but it is slow and requires a human expert. This research uses agents to automatically map database attributes making a distributed database system more robust: a system that recovers quickly from or holds up well under exceptional circumstances. Automated mapping add flexibility by removing the exact match restriction on the JBI.

Section 1.1 discusses the nature of the problem. Section 1.2 outlines the assumptions and limitations of the research. Section 1.3 outlines background information discussing C2 and Joint Vision 2020, and Section 1.4 provides an outline of the rest of the thesis.

1.1 Problem Statement

The objective of this research is to design, implement, and test an automated system for querying multiple data stores in a distributed environment. The sample target environment is the Joint Battlespace Infosphere (JBI).

Current distributed database technology manually integrates data and performs manual conversions. System administrators must manually map database attributes or even write scripts to facilitate interoperability, which takes a lot of time and effort. Thus, changes in the database structure require the database administrator to modify the database mapping in the best case and in the worst case to add additional values to current mappings.

Currently human experts painstakingly create ontologies. The type of ontology proposed herein is a domain ontology in which a human expert generates the domain ontologies in agent classes for specific instances of the objects in the system. This provides a communication language that allows the agents to communicate on the same level. A domain ontology is a mapping between fields of similar content in different databases. Reprogramming, re-instantiation, and retesting are required whenever the domain ontology changes.

1.2 Research Methodology

Agents do not use external help to create the ontology, i.e., data dictionaries, previous training, domain information, or thesauruses. Agents instead use a combination of a string-matching algorithm that compares four aspects of the distributed environment to create the ontology:

1. Attributes of each database in the environment
2. Samples of the data in each database field
3. Database column features
4. The format of the data

This system provides the user with a common interface to diverse database stores thus, eliminating the need to query each data store individually. Moreover, the user requires no information about how many databases exist or how the database stores the information, making the entire system appear as one large database to the user.

In software development, agents must share a domain ontology to communicate. For example, if creating agents to simulate driving a vehicle, many agents need to cooperate to make the vehicle move forward. The agents controlling the steering must communicate with the gas pedal agent, which must communicate with the engine agents. It is possible that these agents cannot communicate with a poorly designed domain ontology; therefore, the vehicle will not be able to move.

Every time a new application is developed, a new ontology and data interface is needed for that application. A discussion of ontologies is in Chapter 2. To integrate the many different systems, multiple manual mappings between these systems must take place to enable data sharing and data communication between them. Every time a user needs to have different data or every time the data environment changes, a manual update of the mappings must occur. There is a risk of complicating the data environment by user error or unnecessary duplication of mappings. Automating this process will save time and labor allowing decision makers access to information without the aid of an expert. Additionally, this would aide human experts in the creation of the ontology whenever a new database comes on-line.

This research implements agents in a distributed system similar to the JBI. The following goals measure the success of agent creation.

- Database wrapping: Provide the ability to wrap information contained in a database into a common environment. The environment should provide mechanisms to publish and serve the information of the database.

- Client subscription: Provide the ability to query and combine services within a common environment without including domain knowledge of the underlying environment structures.

- Agent Autonomy: Provide the capability for agents to act without the need of other agents to complete the task.
- Limited Bandwidth Requirements: Provide the capability for agents to operate in environments where communication is restrictive and still provide ontology creation for the distributed environment.
- Metrics: The automated ontology creation is evaluated by a set of queries. The total number of correct queries is divided by the total number of queries submitted and the percentage is the percent of ontology creation accuracy.

Typically in a distributed database system, each data source stores information in its own way. Database wrapping provides a mechanism to bring these differences together in a format that is consistent with the distributed environment. Therefore, no matter the organization of the data in each individual database, the data access occurs the same way, providing a common environment in which to read and use the data. In this thesis, the database wrapping uses the Java JDBC to ODBC Database Bridge. This bridge provides the agent access to the metadata and data from the database that the agent wraps.

When making a query, the user has no knowledge of where the data originates. The client subscribes to the JBI and queries for information contained within the distributed environment. The query could activate any number of agents depending upon the information each agent maintains in its database. If the agent has the data in question, that agent will return its data to the querying agent. For example, if a client makes a query to provide a list of all instructors and one agent has a column called 'instructor' and another agent has a column called 'professor', both agents will return their data because the domain ontology created by these agents identifies that ‘professor’ means the same as ‘instructor’.

Any agent in the CoABS environment is capable of creating the ontology. It does not matter which agent enters the environment first. What does matter is that one agent must emerge as a leader to conduct the communication and negotiation for ontology creation. In this research, the agents are autonomous and seek a leader when necessary to facilitate ontology creation.

There are locations around the world where networking bandwidth becomes an issue. The ontology creation system will not be effective if it requires a lot of bandwidth to negotiate and distribute the ontology. In fact, it could bring other resources down or restrict some resources from starting. This research chose an algorithm that is small and fast. The agents do not need external resources to negotiate the ontology and the agents send a minimal amount of data for the ontology creation. This keeps communication down and bandwidth available for other resources.

Agents in this research negotiate and formulate an ontology facilitating communication in the environment by using the Jaro similarity metric, discussed in Section 3.3.2. The higher the Jaro metric the more likely the two attributes in question are similar. The agents support the JBI by eliminating the need for manual ontology creation enabling accurate communication.

1.3 Assumptions/Limitations

This research assumes every column in a database table has data in it. Many databases tested have column attributes without data. These columns are deleted for testing. Wrapper agents are assumed to enter the environment before the query agents. The query agents do not deal with ontology creation and leader negotiation, therefore, query agents do not enter the environment until after all wrapper agents have entered and negotiated the ontology.

A limitation of this research is that the databases used for each agent consists of only one table. In addition, this research does not use schema matching or real time data updating.

1.4 Significance

The following sections discuss the current state of C2 and the Chairman of the Joint Chiefs of Staff (CJCS) view on interoperability.

1.4.1 Command and Control

Advancements in technology provide the ability to gather an enormous amount of information to support military operations. Lack of interoperability limits the decision maker's use of this information [2].

Figure 1-1 shows the current state of information gathering in combat systems [2]. Notice all the arrows between the ground and air units go up and down, never horizontal. The lack of horizontal arrows equates to the lack of interoperability between these systems. For example, the Joint Surveillance, Target, and Attack Radar System (JSTARS) must send its information to the ground GSM mobile wireless communications unit to communicate with the Airborne Warning and Control System (AWACS) aircraft. The GSM then sends the information to the CA aircraft, and finally the CA aircraft can send the JSTARS information to AWACS. This process is awkward and time consuming; time not available in a wartime environment.

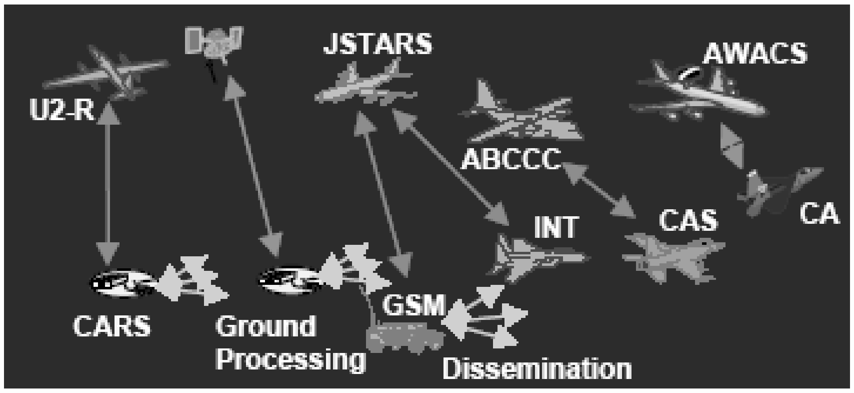


Figure 1-1. Current State of C2 Systems. The arrows between the ground and air units show the lack of interoperability between the air units [3]

The JBI, Figure 1-2, integrates these systems into one cohesive environment enabling communications between any two nodes in the battlespace. In Figure 1-2, the Battlespace Infosphere (BI) integrates planning, command, execution, combat support, and information support into one environment. The BI will serve as an integration system since each function will interact with or be part of the BI while maintaining its own unique required actions [3]. The level of integration with the BI will depend on the information needs of the client and how those needs can be met [3].

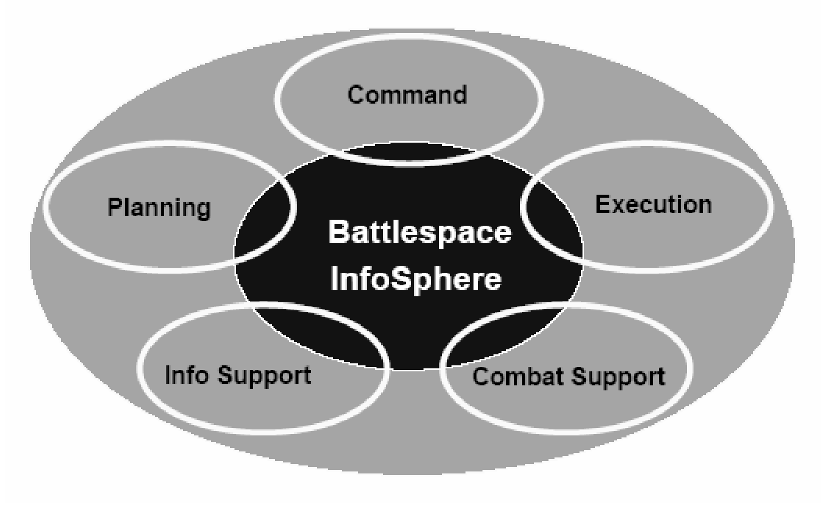


Figure 1-2. JBI information integration [3]

1.4.2 Joint Vision 2020

Joint Vision 2020 is a doctrine that the CJCS distributes throughout the DOD. It envisions how the US military will function in the year 2020 and puts that vision into doctrine.

> "The joint force, because of its flexibility and responsiveness, will remain the key to operational success in the future. The integration of core competencies provided by the individual Services is essential to the joint team, and the employment of the capabilities of the Total Force (active, reserve, guard, and civilian members) increases the options for the commander and complicates the choices for our opponents. To build the most effective force for 2020, we must be fully joint: intellectually, operationally, organizationally, doctrinally, and technically" [4].

This research explores interoperability and demonstrates innovative ideas to ensure that interoperability becomes a reality well before 2020. The CJCS states, "the overall goal is the creation of a force that is dominant across the full spectrum of military operations – persuasive in peace, decisive in war, and preeminent in any form of conflict" [4]. This concept is shown graphically in Figure 1-3.

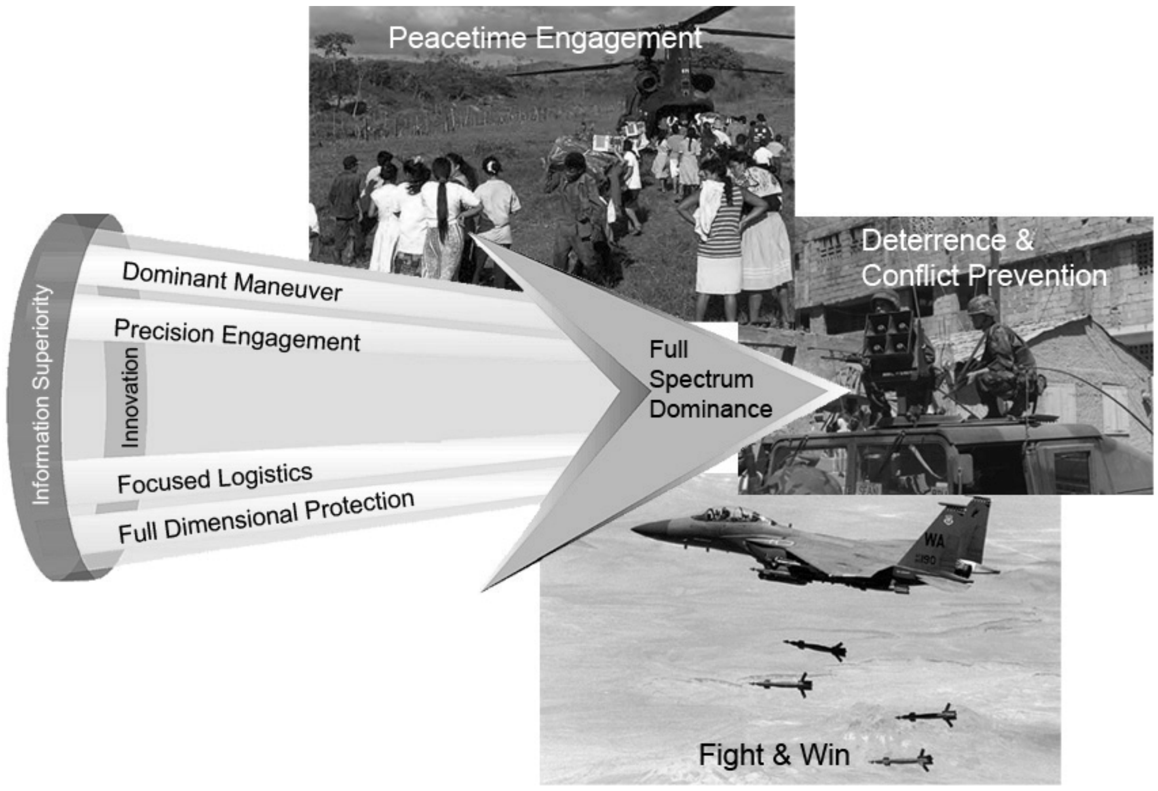


Figure 1-3. Full Spectrum Dominance [4]

During military conflict, the DOD uses many systems to get information. Research is now beginning to examine ways to sift through and organize this information. This will enable decision makers to view data in an organized manner. Currently, a system administrator must select information needed from a list of metadata tags. If the tag is unavailable, there is no information retrieval. In JBI, if an agent registers in the JBI, that agent communicates and coordinates with other agents in the system to retrieve the information requested without any manual intervention.

1.5 Summary

There are many distributed systems used in the DOD today. These distributed systems cannot communicate as one entity without an expert to map the information stores. If the information store changes, experts must adjust the mapping to reflect the change. The manual mapping is called domain ontology creation. The process of domain ontology creation in a distributed environment compounds the force interoperability issue. The JBI provides an environment that combines information stores under one umbrella enabling users to retrieve information from one entity instead of trying to query information from multiple entities. In order for the JBI to work, database administrators must change their data stores to conform to the restrictions of the JBI. This research automates the domain ontology creation by having agents communicate and negotiate the mapping between information stores in a distributed environment. It also helps the JBI by removing some of the restrictions allowing easier information retrieval from the system.

Chapter 2 provides information concerning the domain of the research and compares the paradigms and technologies used in this research with other available paradigms and technologies. Chapter 3 discusses the design and implementation details this research proposes. Chapter 4 evaluates the implementation according to the requirements of Chapter 3. Finally, Chapter 5 concludes the research and describes future work to expand this research.

2. Background

Chapter 2 provides further explanation of the concepts and goals discussed in Chapter 1. Section 2.1.1 details the JBI and the systems used to simulate the environment. Section 2.2 provides information on agents and Sections 2.3 and 2.4 discuss the underlying network used for the JBI. This underlying network consists of Jini^{TM} and CoABS. Section 2.5 covers the concept of ontology, the basis of this research. Section 2.6 discusses data mining and its usefulness in developing domain ontologies, and Section 2.7 covers related work on automated agent ontology creation.

Many approaches have been proposed in the areas of agent and ontology development for application in the Joint Battlespace Infosphere (JBI); however, none of them address the problem of automating domain ontology creation. This thesis expands on previous research [1, 5] by incorporating artificial intelligence techniques into the agents so that agents that register in distributed database environments will coordinate and develop a domain ontology with minimal human intervention.

2.1 Department of Defense (DOD)

The DOD is looking for way to wage war effectively and efficiently with minimal loss of life. To be effective and efficient the right information must be at the right place at the right time. Computer systems and networking bring information to the war fighter. The problem is that there are so many computer systems and so many networks, throughout the DOD, that there is an over abundance of information. To complicate matters, Air Force systems are not compatible with Army systems and information, Army information is not compatible with Navy information, etc. This problem impedes force interoperability.

The importance of force interoperability cannot be overstated.

> "Future military operations will require close coordination and information sharing among heterogeneous units, coalition forces, and other civil and nongovernmental organizations" [6].

The 1990 Gulf War showed the United States' military might and the success of precision guided munitions. This war also demonstrated that the United States military found electronic communication between sister services and coalition forces difficult; interoperability was minimal or nonexistent. Today, the Department of Defense aims to remedy interoperability issues, with the Joint Battlespace Infosphere (JBI). The JBI is the vehicle implementing information sharing and making interoperability a reality, by allowing anyone that registers and connects to be interoperable within the theater of operations The following section discusses the JBI in more detail.

2.1.1 Joint Battlespace Infosphere (JBI)

Putting information in one location does not solve the interoperability issue; however, it is the first step. There is a lot of information that needs to be examined efficiently to make the information valuable. This could ultimately result in putting bombs on target anywhere, anytime.

Two US Air Force Scientific Advisory Board (SAB) reports outline JBI's conceptual framework. Marmelstein [6] summarizes this framework with four key concepts:

- Information is exchanged through publish, subscribe, and query
- Data is transformed into knowledge via fuselets
- Distributed collaboration is achieved through shared, updateable knowledge objects

- Assigned units are incorporated via force templates.

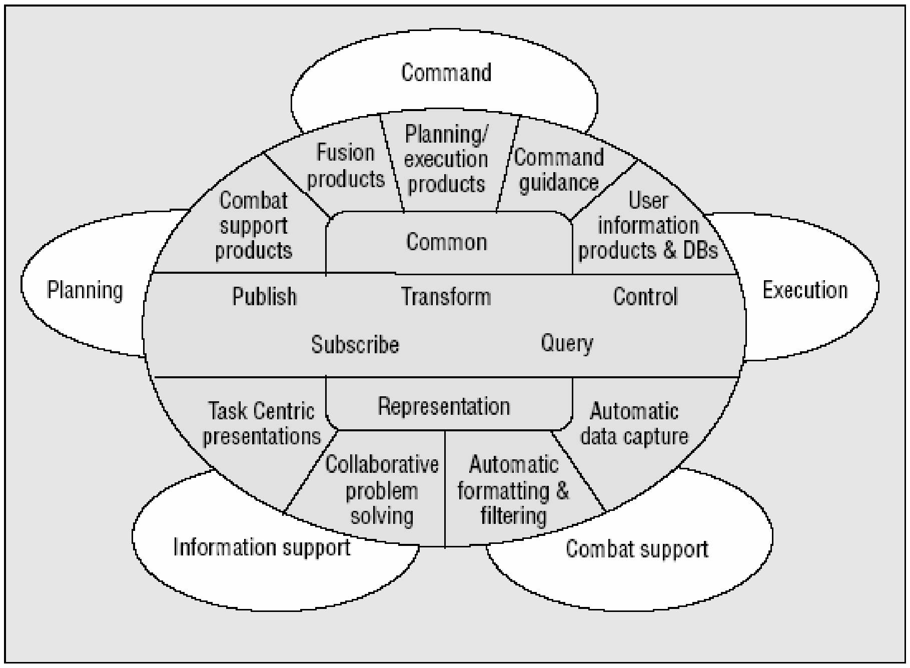


Figure 2-1. The Joint Battlespace Infosphere (JBI) [7].

Figure 2-1 illustrates the interconnections of the JBI with the first key concept being the ability to publish, subscribe, and query, seen in the center controlling all aspects of information gain and information retrieval. Having one system to integrate to enables the military to communicate and exchange useful information with each other. The three core services that the JBI provides are:

- Publish: A client registers itself with JBI and makes available its useful information to any other client in the JBI.
- Subscribe: A client registers itself with the JBI to access information in JBI.
- Query: A client queries JBI for information and perhaps receives what is asked for.

These JBI core services are the foundation for knowledge creation, knowledge exchange, and knowledge retrieval. Additionally, JBI can filter information with fuselets.

Fuselets are compact subroutines designed for a specific purpose or function, such as a searching or computation tool with result returned to the requesting client.

Distributed collaboration through shared, updateable knowledge objects is the third key concept of the JBI. This concept incorporates object oriented programming to create agents (programs) that are able to access, update, and share data with other agents.

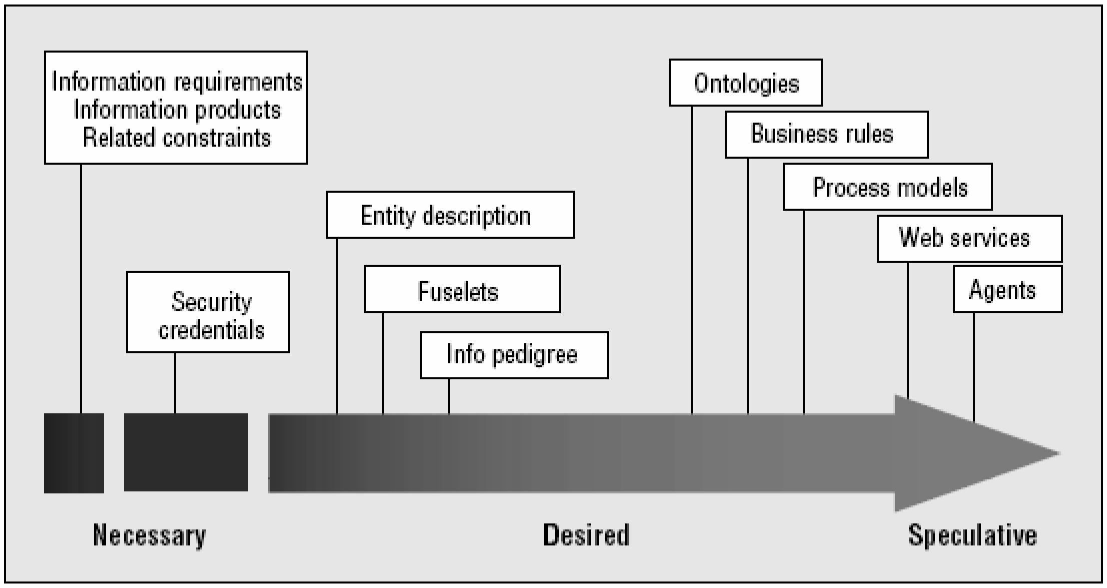


Figure 2-2. Force Template Content [7].

The fourth key concept is unit incorporation via force templates. Force templates allow access to and interaction with other JBI entities. Using force templates enables modularity and the ability to handle content changes which in turn allows JBI to grow or shrink based on the needs of the battlespace. Figure 2-2 depicts force template content. Figure 2-2 also shows where domain ontologies fit in the force template construct.

This research specifically addresses the third key concept of distributed collaboration through shared, updateable knowledge objects. Agents communicate and negotiate formulating an automated domain ontology creating an information-sharing environment from different distributed data sources.

The JBI is currently in its infancy and the Air Force Research Laboratory (AFRL) in Rome, New York, has implemented a test JBI. Using the JBI core services, information flows between clients by sending software objects from one client to another. JBI information consists of two objects, a metadata object, and a published object. Metadata describes the structure and meaning of an object's information and uses for matching a publisher with a subscriber [8].

Before a user publishes an object, a publisher must register it and provide metadata. To retrieve published objects, the subscriber must also register. Publishers and subscribers are matched using metadata attributes and values. A publisher receives metadata attributes and values when it registers. These must match exactly with the attributes and values a subscriber registered. If they do match, the system links them. If they do not match exactly, no linking takes place.

After registering successfully, the client application can publish. The object published can have attributes and values of any type. For example, an XML document, GIF, JPEG, or an ASCII text file. When a publisher and subscriber are registered and matched, the object is published and placed in the subscriber's queue. To use the object, the subscriber must request the object from the queue [8]. After receiving the object, the subscriber can do anything needed with it.

This research focuses on the matching of metadata attributes and values from the publisher with those from the subscriber. Currently the match must be exact. Therefore, if the publisher has a metadata attribute "Windspeed" and the subscriber requested a metadata attribute "windspeed", a match would not occur even though the words are the same and differ by a capital letter. JBI is too restrictive with the requirements to operate in a contingency environment. The next section gives an overview of a program agent.

2.2 Agents

Russell and Norvig [9] define an agent to be “an encapsulated computer system that is situated in some environment and is capable of flexible, autonomous action in that environment, in order to meet its design objectives”. An agent is a program that performs some information gathering or processing task to meet defined goals. Agents also provide a mechanism for integrating multiple software systems.

There are two classifications of agents: weak and strong. Weak agents are autonomous, social, reactive, and proactive. Autonomy means that agents can act on their own. Social agents are able to interact with each other. Reactive agents respond to stimulus and agent pro-activity means agents take initiative. Strong agents have all the characteristics of weak agents but are also mobile, veracious, benevolent, and rational. Mobile agents can move. Agents having veracity are agents that are truthful, benevolent agents do what they are told. When agents are rational, the agents will perform purposefully to achieve goals.

Agents are powerful programming entities useful in a wide variety of areas. Agents are used to simulate two aspects of the JBI. The first agent is the database agent. The database agent simulates the JBI data repository that holds all of the information currently in the JBI. The second is the query agent and the query agent simulates a client querying the JBI system to retrieve information.

The next two sections describe Jini^{TM} and CoABS, two applications that handle agent discovery and agent-to-agent communication.

2.3 Jini^{TM}

Sun Microsystems released Jini^{TM} technology in 1999 as a platform for building applications with knowledge of the resources of their underlying network. The Jini^{TM} architecture provides an agent the ability to announce itself to the network, provide some details about its capabilities, and immediately become accessible to other agents in the network environment.

Jini^{TM} provides a reliable network interface so services can join and leave a network. A reliable network has the ability to continue operating in the event of a system failure with little impact to the user. If a service crashes, the client locates another or waits for the initial service to reappear. If a communications link is lost, the client and service renegotiate another [2]. All of this happens without user intervention.

A Jini^{TM} network is scalable and secure. It has no central control; allowing networks to manage themselves [2]. Users can add and remove services without the need for a central entity to coordinate. In addition, Jini^{TM} maintains the integrity of network look-up tables. Dynamic discovery of Jini^{TM} services make this happen.

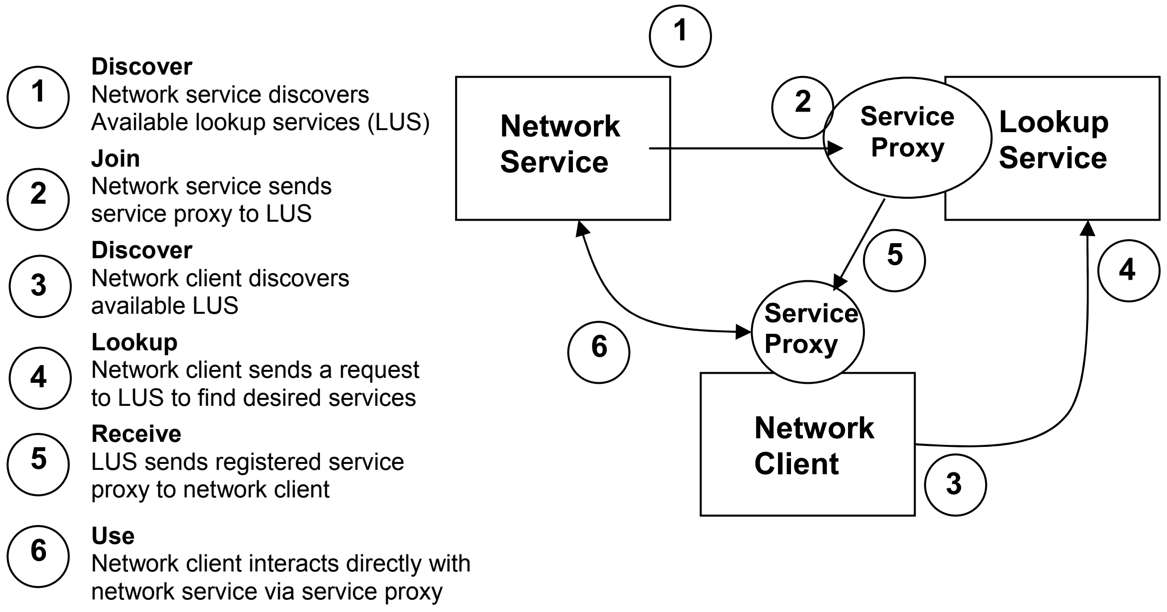

1
Discover
Network service discovers
Available lookup services (LUS)
2
Join
Network service sends
service proxy to LUS
3
Discover
Network client discovers
available LUS
4
Lookup
Network client sends a request
to LUS to find desired services
5
Receive
LUS sends registered service
proxy to network client
6
Use
Network client interacts directly with
network service via service proxy
Network Service
Service Proxy
Lookup Service
Service Proxy
Network Client

Figure 2-3. How Jini^{TM} Technology Works [10].

Communication across a Jini^{TM} network is a six-step process as shown in Figure 2-3. The first two steps of Jini^{TM} communication comprise the registration portion of Jini^{TM}. During the registration process, a service sends a service proxy to all lookup services (LUS) on the network, or to a selected subset [10]. Services are the cornerstone of a Jini^{TM} network and use lookup services to advertise their capabilities.

Steps 3 and 4 comprise the discovery portion of Jini^{TM}. Discovery occurs when a requesting service locates a LUS and asks for a registered service. If the LUS does not have the service, the service requester locates another LUS [2].

The fifth and sixth steps are the communication portion. If the LUS has the service requested, it returns the proxy of that service to the requester [2]. From here on, the proxy brokers all communication between the client and the service.

Jini^{TM} Network Technology provides a powerful infrastructure that allows services to interact with little foreknowledge of location or underlying network. It offers a highly scalable solution to the problems of network transport [7].

To make this networking and agent-to-agent communications complete, however, some middleware services are required.

2.4 Control of Agent Based Systems (CoABS)

CoABS is a framework built on top of Jini^{TM} [5]. CoABS supports the seamless integration of agent-based systems. Like Jini^{TM}, CoABS provides a scalable and flexible environment for systems to participate in, Figure 2-4.

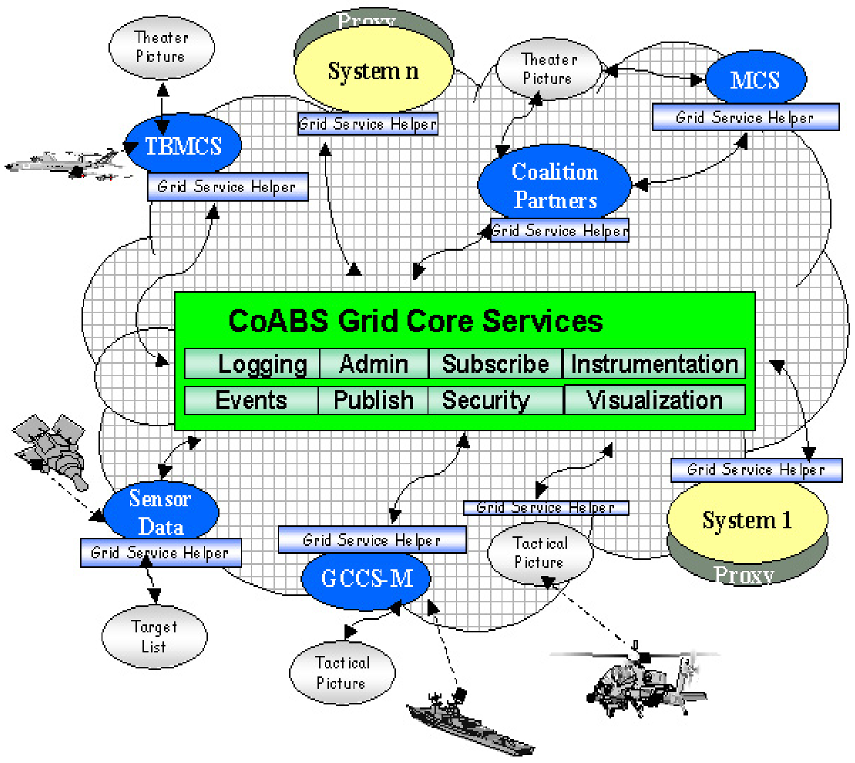


Figure 2-4. CoABS Grid [11].

The CoABS grid includes a method-based application programming interface to register agents, advertise their capabilities, discover agents based on their capabilities, and send messages between agents. The Grid also provides a logging service to log both message traffic and other information; a security service to provide authentication, encryption, and secure communication; and event notification when agents register, deregister, or change their advertised attributes [11].

Software agents use CoABS to register within a common distributed environment gaining access to several services that help them communicate. One of these services is a LUS, which provides agent discovery services [1]. Along with registration and the LUS, CoABS features an agent messaging system. This messaging service provides a transport mechanism to deliver messages between agents [1].

CoABS also has a Graphical User Interface (GUI) that allows an administrator to manage and monitor the CoABS grid [11]. This grid is the front end to three daemons, HTTP daemon, LUS daemon, and Remote Method Invocation (RMI) daemon. These daemons provide the necessary services for agents to advertise their capabilities and solicit capabilities from other agents [11]. Since CoABS is built upon the Jini^{TM} Network Technology, CoABS utilizes Jini^{TM}'s six-step process to facilitate clients and services to connect and communicate.

While the CoABS grid is up and running, Jini^{TM} is working transparently in the background. These two applications work in tandem, creating an ideal environment for agent negotiation and automated ontology creation.

2.5 Ontology

The field of Ontology studies the nature of existence. Applying this definition to agents: “an ontology is a description, like a formal specification of a program, of the concepts and relationships that can exist for an agent or a community of agents” [12]. In software development, an ontology implies agents that communicate, communicate on the same level. Every agent in the system knows how to communicate and understand the messages communicated.

During the initial design of a multiagent system, an ontology is user-defined. With an ontology in place, agents in the system have a vehicle for communication and each agent understands the language communicated. The programmer implements an ontology within an agent so agents can perform and achieve their goal. Agents using ontologies can increase the efficiency and effectiveness of current Internet services thereby reducing human intervention [5]. If multiagent systems could effectively communicate with each other via an ontology, human intervention could be largely eliminated. A user would simply submit a query and obtain results.

One type of ontology is domain ontology. Domain ontologies define all concepts and relationships in a specific domain [13]. Developing a complete domain ontology takes time. In addition, reprogramming, re-instantiation, and retesting are required when an ontology requires a change. Problems arise from manual domain ontologies because the domain expert cannot define everything there is to know in a specific domain. Over time, things change and evolve, and the multiagent system will only be as good as the expert. If the expert does not know something about the domain, then the agents will not either. If agents could create, modify, and apply their own ontology, a multiagent system could adapt, grow, and shrink autonomously.

2.5.1 Multiagent Systems Engineering (MASE)

Tools for automating agent creation in multiagent systems are in development. These tools make agent creation easier but the user must know the system well to take advantage of the tools. Ontology creation is one-step of this process and is currently a manual process. DiLeo [13] takes the reader through the process of manual ontology creation from the software engineer's point of view. The steps for manual ontology creation are:

1. Define purpose and scope.
2. Collect and analyze data.
3. Construct the initial ontology.
4. Refine and validate the ontology.

The next few paragraphs discuss these steps in detail.

A lot of time and thought must go into an effective ontology design. It is an evolutionary software engineering process. DiLeo argues engineers developing software systems using agents should allow equal design time to the agents and the ontology because once you release the agents into the system, agents do what they are told and adapt if the environment changes. If agents are to perform consistently and provide the same expected output no matter how the environment changes, the agents need a domain ontology. In addition, the domain ontology must be adaptable. With DiLeo's process every time something changes in the environment, a manual change to the domain ontology must take place.

A designer must describe ontology requirements as well as the range of intended users of the ontology [13]. For example, when designing a multiagent system to perform course scheduling, the ontology must define abstract classes regarding courses, quarters, instructors, and classrooms. The software requirements and the goal hierarchy help define the ontology's purpose. The purpose describes why the ontology exists, such as to list all classes in the education domain required when scheduling courses. The scope defines to what level of detail that an ontology describes the objects, such as defining only the semantic ideas necessary to schedule courses in a distributed network environment [13].

After defining the scope, the designer can build the model. The designer creates a list of possible terms or concepts that the ontology must describe. Designers form this list by examining the goal hierarchy, use cases, and sequence diagrams from the previous MaSE steps for candidate ontology terms [13]. Actions in a sequence diagram show which terms could be part of the information passed in the system. The designer examines the system requirements, goal hierarchy, and use case models in a similar manner to create the candidate term list for the ontology [13].

To construct the initial ontology, the term list is organized into classes and attributes, and an initial draft of the data model is produced. Before creating an entirely new ontology, a designer must determine whether any existing ontologies can meet the system needs. If no existing ontologies fully specify the information needed for the system, the designer must build a new ontology [13].

At this point, a designer must refine and validate the ontology. A designer reviews use cases and sequence diagrams to ensure the information specified in the ontology accurately reflects system events. If not, the designer makes corrections until the ontology accurately reflects system events.

This brief overview of the manual process of ontology creation illustrates the labor-intensive nature of the process. This research differs from DiLeo's in that it provides an implementation of an agent created ontology system so that the ontology can change as the environment changes without the human intervention. A system where if the environment changes, the agent ontology also changes—keeping everything in lock step.

The previous sections discussed the importance of the JBI to the DOD, the JBI and how it works, the underlying network of JiniTM and CoABS, and ontology creation in multiagent systems. Before trying to program agents to negotiate ontologies, the focus turns to data mining concepts. An application of data mining is extracting general concepts from relational databases. The next section discusses data mining and how data mining is used in this work

2.6 Data Mining

Data mining is the process of extracting knowledge from a large amount of data [14]. Several data mining approaches exist and, as with most design problems, the application dictates the approach. There has been a wealth of research in databases making use of data mining techniques to merge many databases into a single unit. This research makes use of some data mining techniques, specifically, cluster analysis. We discuss cluster analysis in detail in Section 2.6.2.

2.6.1 Database Record Matching

There are many techniques to implement record matching in databases. Record matching or name matching has been explored in many fields; statistics, database, and AI. In statistics, the problem is called probabilistic record linkage [15]. Probabilistic record linkage allows the assembling of information from different data sources [16]. Record linkage is the process of finding a unified record from two or more records that are in different files but belonging to the same entity. Probabilistic linkage takes into account the uncertainty that exists in comparing variables used for comparison in both files [16].

Knowledge intensive approaches provide the basis for record matching in the database community [15]. Finally, in AI, learning the parameters of string distance metrics and combining the results of different distance functions uses supervised learning [15]. Three types of distance functions used are:

1. Edit-distance like functions
2. Token-based distance functions
3. Hybrid distance functions.

All three distance functions listed above map a pair of strings *s* and *t* to a real number *r*, where a smaller value of *r* indicates greater similarity between *s* and *t* [15]. The edit distance functions represent distance as the cost of the best sequence of edit operations that convert *s* to *t*. Edit operations include character insertion, deletion, and substitution, and the function assigns each operation a cost. Token-based distance functions are those that operate on groups of characters (tokens). Hybrid distance functions are those that use pruning methods to reduce the set of string pair comparisons.

Since database agents will be working in a distributed environment, the sizes of the databases are potentially unknown so the Jaro similarity metric was used. The Jaro similarity metric is not an edit distance metric but it provides effective results with a minimal amount of computation. Section 3.3.2 discusses the Jaro similarity metric in detail.

The Boyer-Moore string-matching algorithm [17] was considered. Implementations of text editors use this algorithm more frequently for search and inserts. Boyer-Moore is more efficient as the search string gets longer. In this application, the string compares are generally short, so the Boyer-Moore algorithm was not used. The Jaro similarity metric is efficient and works best with short strings, and provides us with a numerical matching, and the substrings for storage.

This thesis combines techniques from all three data-mining areas, statistics, to measure the success of automatic agent ontology creation; databases, implement clustering to analyze the data and group similar data objects; and AI, to determine the degree of similarity between two data objects. These agents will accomplish this task without the use of external word matching sources, like data dictionaries, previous training, domain information, or thesauruses.

2.6.2 Cluster Analysis

Clustering is a method of grouping objects into classes by some metric of similarity [14]. Clustering sifts through large data stores, grouping related objects together into a cluster. Clustering uses statistical techniques like mean and variance, to manipulate the clusters extracting information about the data so the data provides meaning to the user. The difficulty in clustering is finding the right balance of technique and speed so that data searches take a minimal amount of time.

The database agents are implemented as wrapper agents. Wrapper agents have access to all of a database's information. When clustering, two agents communicate and decide whether they have matching fields. Agents calculate cluster information on their database and then communicate these metrics to decide whether a match exists. Agents create a logical link between the fields in the database.

Several authors have explored the intelligent database querying and ontology creation. In the following section, we discuss their work and analyze how it relates to this research.

2.7 Related Work

The benefit of the research described herein consists of relieving a programmer from generating all metadata mappings from one database to another. Agents are able to set up their own mapping and the programmer will only have to monitor the progress or adjust mappings as needed.

Name matching research has been on going for years and is still relevant today. Researchers have run experiments using applications such as WordNet, described below, to improve name matching success rates. Section 2.7.2 discusses RETSINA. Sections 2.7.3 and 2.7.4 discuss two other projects that have implemented name matching, DELTA and SemInt, respectively. Lastly, there is a discussion of different algorithms showing correct name matching results.

2.7.1 WordNet

WordNet is an application to ease syntax learning in a given project. WordNet organizes nouns, verbs, adjectives and adverbs into synonym sets, each representing an underlying lexicographical concept [18]. Different relations link the synonym sets, and provides a way to match words from one data source to other data sources. In applications requiring a method to decide whether words are topically similar, WordNet can make that distinction. WordNet requires a programmer to provide the ontology for the application domain. In addition, since WordNet is a separate program, the application must have a place in your distributed environment. Each time there is a call to WordNet to check for a match, the communication required uses precious bandwidth. Depending upon how many WordNet accesses there are, it could be a time intensive task and could affect network performance.

JBI is a distributed environment that, depending on the contingency location, can possibly be a restrictive, low bandwidth network with minimal services. Further, using an application like WordNet may not be conducive to wartime operations. This research uses agents combined with data mining techniques to negotiate an ontology in whatever application the agents are running in. It eliminates the need for an external translation service and it enhances the JBI for worldwide implementation.

2.7.2 Artificial Intelligence

There are two principle categories of learning in multi-agent systems: centralized and decentralized. In centralized learning, a single agent executes the entire the learning process and does not require any interaction with other agents [19]. Decentralized learning has more than one agent engaged in the same learning process. Negotiation and cooperation is mandatory for successful decentralized learning. Given the distributed nature of the agents, centralized learning is not feasible, and this thesis focuses on decentralized learning where multiple agents communicate and coordinate toward a common goal.

The following are a few examples of decentralized learning of agent communication in different environments. A discussion of Reusable Environment for Task-Structured Intelligent Networked Agents (RETSINA) [20] and its significance is in the next section.

2.7.2.1 Reusable Environment for Task-Structured Intelligent Networked Agents (RETSINA)

RETSINA is an open multi-agent system (MAS) that supports communities of heterogeneous agents. The RETSINA system agents form a community of peers that engage in peer-to-peer interactions. Any coordination structure in the community of agents should emerge from the relations between agents, instead of the imposed constraints of the infrastructure itself. In accordance with this premise, RETSINA does not have a centralized control within the MAS; rather, it implements distributed infrastructural services that facilitate the interactions between agents, as opposed to managing them [21].

RETSINA most closely resembles the JBI concept. The JBI implements concepts from RETSINA such as no centralized control within the MAS and it allows CoABS to monitor the agent-to-agent communication.

The next sections discuss successful applications designed for name matching. These applications are useful for agent ontology creation, but have positives and negatives associated with them.

2.7.3 Data Element Tool-Based Analysis (DELTA)

DELTA is a tool that uses textual similarities between data element definitions to find candidate attribute correspondences [22]. A commercial full-text information retrieval tool (Personal Librarian) is used to search and find attribute matches. DELTA first gets all available metadata for an attribute and saves that information as a text document. A human, called the integrator, has to choose the attribute and the database to perform the match.

Querying the Personal Librarian is a way to find attributes. The query returns documents ranked using a weighted similarity of terms. The search for corresponding attributes in the list is a manual search. The default search pattern is the full text of a metadata document in one of the databases [22]. In addition, query searches are applied to all the words in a document. This allows finding matches even where the attribute names are very different but where there are similarities in the definition. For example, searching for delivery address found the proper attribute from two different data dictionaries and the attribute names were different [22].

Control over the order of tasks is manual [22]. This provides tool flexibility, but also requires manpower to make DELTA useful. This research proposes intelligent agents respond to a query on their own without a human controller. The agents query each other to decide which attributes are the same freeing the analyst to examine the data from the query.

The data used to test DELTA comes from three databases. The first, Advanced Planning System (APS) is a relational database that has 884 attributes in 107 tables with a dictionary with 739 elements. The second database, Computer Aided Force Management System (CAFMS) is another relational database that consists of 1056 attributes in 162 tables with 637 data elements, and no data dictionary. The third database, Wing Command and Control System (WCCS), has 2578 attributes in 294 tables with 1760 data elements. Because no data dictionary was available for the CAFMS database, DELTA was not tested with it. Identifying the correct attribute in the table involves a manual search. The authors state that manual search takes approximately 15 minutes per attribute. DELTA achieved a 40% match success rate. Without perfect knowledge and a perfect data dictionary, DELTA is not a good choice for database attribute matching. The risk is that people make mistakes, which cannot be tolerated during wartime.

SemInt is an extension to DELTA that differs in two ways. First, SemInt does not need perfect knowledge to provide good results and second, SemInt uses a neural network instead of the personal librarian.

2.7.4 SemInt

SemInt is an automated system for determining candidate attribute correspondences [22]. SemInt differs from DELTA by providing good results when databases are not well documented and a client does not have perfect domain knowledge.

SemInt generates 20 numeric properties from the metadata and population for each attribute. It then determines which properties are most useful for discriminating among attributes and produces a classifier function using a back-propagation neural network. The metadata used includes data types, length, keys, foreign keys, range constraints, and access restrictions. Population information used includes average, maximum, and variance for numeric values. For the text field values, string length statistics are used. The authors call these calculated items discriminators, and are used to determine attribute matches. The network needs training before it can be useful, because SemInt uses a neural network. For large databases, this training can take from a few hours up to days of CPU time to complete. In a distributed wartime environment where execution time is paramount, this is an unacceptable risk; as foreknowledge of data requirements needed is required.

SemInt operates in a two-phase process. The first phase is the training of the network on a database chosen as the reference database. The human integrator provides the classification for the training of the network. SemInt computes the discriminator vector. The second phase uses the neural network to map attributes of other databases onto attributes of the first database effectively comparing attributes from the foreign database with those in the reference database. Discriminator computations use the foreign attributes and the classifier applies to a discriminator vector. The vector and each cluster in the reference database returns a similarity. Once all attributes are processed, SemInt returns a list of similarities. The high valued attributes for each cluster are the suggested correspondences for that cluster [22].

A correspondence is the relationship between two attributes. Put another way, a correspondence means 'means the same'. The best average number of candidate correspondences SemInt found for each attribute is 43% out of 50 correspondences identified. The recall percentage is the number of correspondences found divided by the total number of correspondences identified [22]. SemInt's best recall percentage is 44%. Thus, by itself, SemInt is not adequate. However, it does not require domain knowledge or a database with a data dictionary to provide good results. On the other hand, the authors claim that SemInt along with techniques from DELTA may provide results with less human intervention to determine attribute matches.

This research is different from SemInt in many ways. Database agents learn the domain ontology with string matching techniques and store that knowledge in a hash table. This research does not require a neural network, which means no time spent training a neural network (which for SemInt, could take days). Further, it then is not restricted to the concepts capable of being learned by a neural network.

2.7.5 Other Related Works

Another system, Automatch uses Bayesian learning techniques. The system acquires probabilistic knowledge from domain experts stored in an attribute dictionary [23]. Automatch uses the attribute dictionary to find optimal matching. Using cross validation, Automatch achieved a match rate of 66%. In a separate experiment, and using random guessing to generate the same attribute matching pairs, Automatch only achieved a match rate of 10%.

COMA combines multiple matches in a flexible way. The match operation takes as input two schemas and determines a mapping indicating which elements of the input schemes logically correspond to each other. COMA can make use of results from previous match operations. Without reuse, the single-matchers' average precision is 50% and the average recall is 81%. The average overall match rate is no more than 45% [24].

Cupid considers two types of matches, a thesaurus for linguistic matching, and a structured matching based upon similarity of contexts or vicinities. Cupid does well using the thesaurus and very poorly without the thesaurus in some instances. No measurable results were provided [25].

Madhaven, et al. [26] introduces the concept of corpus based schema matching. They save previous matchings in a component called the mapping knowledge base (MKB). The MKB learns classifiers for each of the schema elements seen in the past. When the classifier for an element *e* is applied to a new schema element *e1*, it predicts the degree of similarity between *e* and *e1* [26]. Coupled with the MKB, the authors use five different base learners trained to recognize each type of element the MKB captured.

The authors compared their MKB matching with a basic matcher. The basic matcher uses the five base learners, but the training is only on the matching attributes. The basic matcher achieved between a 65% and 85% recall accuracy. The MKB achieved between 72% and 84% accuracy and the combination of the two achieved between 78% and 90% recall accuracy.

Paolo Bouquet, et al. proposes a new algorithm to use for schema matching [27]. They address the problem of deducing relations between sets of logical formulae that represent the meaning of concepts belonging to different classification. The matching consists of three knowledge parts:

- Lexical: knowledge about the words used in the labels
- Domain: knowledge about the relation between the senses of labels in the real world or in a specific domain
- Structural knowledge: knowledge derived from how labels are arranged in a given hierarchical classification (HC) [27].

Lexical and domain knowledge is not used to improve the results of structural matching. Instead, knowledge from all three levels is used to build a new representation of the problem, where a logical formula represents the meaning of each node and relevant domain knowledge and structural relations between nodes are added to nodes as sets of axioms that capture background knowledge [27].

Once the meaning of each node, together with all relevant domain and structural knowledge is encoded as a set of logical formulae, the problem of discovering the semantic relation between two nodes becomes a simple problem of logical deduction [27].

The algorithm used is CTXMatch, which takes two HCs and returns a set of mappings between their nodes. WordNet provides both lexical and domain knowledge [27]. Results show the percentages of accuracy and recall between the two search domains of architecture and medicine with the Google and Yahoo! search engines. The best equivalence achieved was 78% precision with a recall of only 13%. The architecture search did a bit worse with 71% precision and 10% recall.

Authors Williams and Tsatsoulis research matching diverse ontologies using concept cluster integration [28]. Each agent has their own ontology and tries to discover relationships between themselves if one exists. For example, if one agent's ontology was 'NBA' and the second agent's ontology was 'College Hoops', the two agents should discover their relationship 'basketball' [28]. The authors' agents use supervised inductive learning to learn their individual ontologies. Only 20% of the queries produced verified concept cluster relations.

The final system discussed is the Learning Source Descriptions (LSD) system [29]. LSD employs and extends current machine learning techniques to find attribute matches, semi-automatically. The user must provide semantic mappings for a small set of data sources, ‘small’ being undefined. LSD uses these mappings together with the sources to train a set of learners. After training the learners, LSD finds semantic mappings for a new data source by applying the learners and combining their predictions using a meta-learner [29]. The results are very good using this technique. LSD achieves high accuracy from 71% to 92%.

Do, Melnik, and Rahm suggest that to identify a solution for a particular match, it is important to understand which of the proposed techniques performs best, i.e., effectively reduce the amount of manual work required for the match task [30]. To show the effectiveness of their system, they demonstrate its application to some real world scenarios [30]. The system evaluations were done using diverse methodologies, metrics, and data making it difficult to assess the effectiveness of each single system. Furthermore, the systems are primarily not publicly available making it virtually impossible to apply them to a common test problem or benchmark in order to obtain a direct quantitative comparison [30].

Four different comparison criteria were used: input, output, quality measures, and effort. Input considers schema languages, relational, XML, etc; number of schemas and match tasks; schema information; schema similarity; and additional information used, i.e. data dictionaries or thesaurus’ used to help facilitate the matching. The output considers which elements correspond to each other. The quality measures used are the results compared to a manual match. To assess the manual effort they consider both the pre-match effort required before an automatic matcher can run as well as the post-match effort to add the false negatives to and to remove the false positives from the final match result [30]. They review the evaluations of eight different match prototypes, Autoplex, Automatch, COMA, Cupid, LSD, GLUE, SemInt, and SF.

The results show that the best match quality range from under 20% to over 90%, but the way the systems have been tested varies to a great extent from evaluation to evaluation [30].

Yatskevich [31] completes another evaluation of different schema matching algorithms between Similarity Flooding (SF), COMA, and Cupid. Yatskevich states that there are five different classifications of attribute-matching approaches:

- Hybrid or composite. Hybrid matcher uses multiple criteria to obtain mapping. A composite matcher combines results obtained by exploiting several matching algorithms.

- Weak or strong semantics. Weak semantic techniques includes string, data type and soundex analysis. Strong semantic techniques use precompiled thesaurus and dictionaries.

- Instance based or schema based. Instance based matchers consider data instances. Schema based matchers rely only on schema level information.

- Element or structure level. Element level matching is performed to individual elements. Structure level matchers consider combinations of elements such as complex schema structures.

- Language or constrained based. Language based matchers use a linguistic approach like comparing names of element *s*. Constrained based matchers exploit constraint information i.e., relations, keys.

Weak semantic schema based matchers represent both hybrid and composite approaches for the evaluation. Various element and structure level techniques of analysis language and constrained based information. Comparing all three matching algorithms, SF, COMA, and Cupid, the best precision achieved was 84%. However, overall precision was significantly worse at 30%.

All of the approaches described above are attempts at automating ontology creation. Table 2-1 summaries these approaches and gives a picture of how each technique performs, giving the best match percentage and what the technique uses in order to achieve that best match percentage.

Table 2-1. Comparison of Different Approaches

Approach	Technique Used	Level of user input required	Outside Sources	Amount of Communication	Best match percentage
DELTA	Manual Searching for corresponding attributes	A human chooses the attribute and the database to perform the match.	Personal Librarian	Name/Information/ entire field contents	40% match success rate.
SemInt	Back-propagation neural network	User provides the training data	None	Name/Information/ Statistics	43% match success rate
Automatch	Bayesian Learning	Knowledge from domain experts	Attribute Dictionary	Name	Using cross validation: 66%, otherwise only 10%
COMA	Schema Matching	Provides match and mismatch information	Previous Matches	Name/Information	No reuse, Average overall is no more than 45% match rate
Cupid	Linguistic and structural matching	None	Thesaurus	Name/Statistics	No Quantitative Results
MKB Matching	Learns classifiers for each of the schema elements seen in the past	None	Previous Schema Matches	Name/Information/ Statistics/entire field contents	84%-72% match success rate
CTXMatch	Hierarchical Classifications	None	WordNet	Name/Information	78% match success rate
Concept Cluster Integration	Supervised inductive learning	None	None	Name/Information/ Statistics/entire field contents	20% match success rate
LSD	Machine Learning	User provides semantic mappings for a small set of data sources	None	Name/Information/ Statistics	92%-71% match success rate

This thesis' best match result is a 79% agent response rate. As can be seen in the comparison these results are competitive to the best systems while requiring much less user input and prior training. For those systems that do not require these elements (COMA, Cupid, etc) this research outperforms them significantly.

The 'Amount of Communication' column identifies the particular technique used to obtain the results. For example, in order for DELTA to achieve its 40% match success rate, DELTA uses the attribute name, attribute information (data type, etc), and it searches the entire column contents to determine its match. CTXMatch only needs the attribute name and attribute information to achieve its 78% match success rate, but it uses WordNet to help with its attribute matches.

2.8 Background Summary

The automatic generation of domain ontologies assists with the DOD's interest in creating an interoperable environment for information sharing. Jini^{TM} and CoABS provide the networking backbone that emulate the functionality of the JBI and make agent communication easy, flexible, and secure. In this environment, agents facilitate their own communication and as a group negotiate a domain ontology with minimal human intervention.

Many of the implementations discussed in this chapter provide background into agent ontology creation applicable in a distributed database environment. However, several methods require overhead or manpower that in a JBI like distributed wartime environment is not feasible. This research overcomes these limitations by eliminating user and external information to create the corresponding ontology all while maintaining match accuracy.

DELTA and SemInt are tools that attempt attribute matching across heterogeneous databases. DELTA requires perfect knowledge and data dictionaries in order to be successful. SemInt is an improvement over DELTA in that it does not need perfect knowledge, and user interaction, and can attribute match with good results. SemInt uses a neural network that requires training before it can be of any use. Depending on the training set, this training can take hours or even days to complete. The advantage is that computations are fast once training is complete for the network. The advantage to this research is that like SemInt, there is no need for perfect knowledge or data dictionaries. In addition, this research does not use a neural network because of the complexity and overhead of using the technique.

As in the related work, this implementation seeks to provide a method of finding matches between two database fields without comparing the entire set of records. Unlike these approaches, this research attempts matches based only on the data and the data's attribute information, through substring statistics. Even so, incorporating the features used in SemInt, and the other systems into this work could possibly produce more accurate ontology mappings, as will be discussed in the results section, after discussing the implementation details in the following chapter.

3. Design and Implementation

Many researchers have tried to solve the attribute matching problem using many different techniques or combinations of techniques in order to obtain good results [15, 22, 23, 24, 25, 27, and 29] as discussed in the previous chapter. This chapter discusses how the goals in Chapter 1 are met through database wrapping, client subscription, agent autonomy, and limited bandwidth requirements.

In addition to these goals, this research considers two other goals. The first goal is simulation of the JBI environment. The agent environment should provide the most accurate test environment possible. This will ease porting this research agent code into the current JBI.

The second goal is domain ontology creation with minimal communication. This goal has agents create a specific ontology, one that depends upon the data inside the JBI, allowing services within the environment to find and communicate with one another. The number of successful query returns given a set number of queries made measures the success of ontology creation.

This chapter outlines the procedure on how this research automates ontology creation. This not only aids efforts in the JBI development, but also assists any organization needing to merge data from different sources in a distributed environment without manual data mappings.

This chapter is organized as follows. Section 3.1 discusses the requirements for the research. Section 3.2 details the architecture of how the agents integrate into a distributed environment. Section 3.3 discusses database wrapping and the agent implementation, respectively. Section 3.4 defines the query process and Section 3.5 provides the summary of the chapter.

3.1 Requirements

For agent-to-agent communication to take place, the agents must be in a distributed environment. Jini^{TM} and COABS provide the distributed environment where agents can communicate quite easily. Once CoABS is successfully operational, the agents enter and run in this environment.

Creation of the domain ontology begins by matching agents' metadata. If the metadata do not match, the agents examine a sample of the data in the database. Using the Jaro similarity metric a determination of match is calculated. Once updates to the domain ontology are complete with the metadata, the lead agent distributes the updated domain ontology to all agents in the system.

3.2 Architecture

In the JBI, data sources vary as does the information contained therein. There is aircraft information, weather, weapons, locations, targets, and so on. To publish to the JBI, the client must conform to a rigid set of rules. This helps keep the JBI in a uniform, consistent state. XML takes care of the matching, but one big restriction of XML is that the match must be exact. If a client wishing to make a query does not know how or what to query, or does not know the exact spelling of the query, it is likely the client will not retrieve any information from JBI at all.

This research creates database wrapper agents that access the data contained within the databases. These agents communicate, negotiate, and create a domain ontology without any help from an outside human expert or application. The agents themselves determine if ‘professor’ matches ‘instructor’. When a query comes in asking for professor, both agents respond; the client does not have to make another query for instructor.

Figure 3-1 shows how agents operate in the CoABS environment. Explanations of the elements of the figure are below.

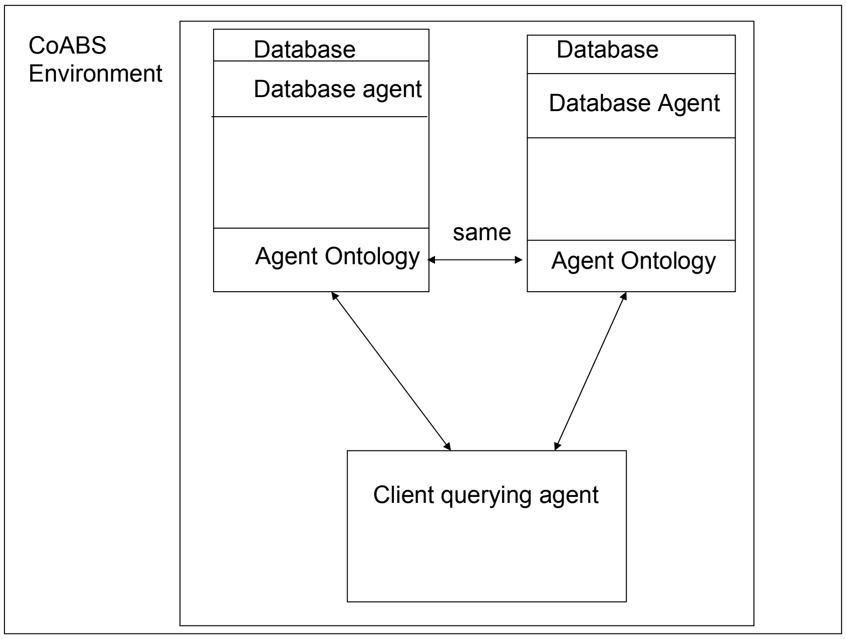


Figure 3-1. Agents in CoABS Overview

3.2.1 Agents

As shown in Figure 3-1, there are two types of agents the database agents and the query agents. Section 3.2.1.1 discusses the details of the database agents including how they create the domain ontology. The domain ontology is created when the second agent enters the system. This second agent transmits its database information to the lead agent where the lead agent invokes the Jaro similarity metric method, updates the ontology, and distributes the updated ontology to all agents in the environment. Section 3.2.1.2 discusses the details of the query agents including how they register in the CoABS environment and how they query the wrapper agents to get the information they request.

3.2.1.1 Database Agent

When the database agent wraps itself around this repository, the agent has access to and provides all of the data contained within if queried. The agent also knows all of the metadata about the database, for example, data types, and field lengths. Upon wrapping, the agent stores all of the repository attributes into a vector. It pulls random data from each field, depending upon the size of the database, to develop keyword vectors for each field. The keyword vector holds information on the commonality found within each attribute column of the database. For example, suppose a column in the database was 'instructor'. The agent looks at random data elements of 'instructor' and sees if there is commonality in the data; the data may all start with 'Dr.' or 'Prof'. The agent finds these common elements and stores them in a keyword vector. Creating a keyword vector eliminates the need to send all of the agent's database data to the lead agent. This keeps communication and data transmission to a minimum while still providing acceptable results.

The keyword vector is created using part of the Jaro similarity metric method with samples of the data contained inside the databases. The keyword vector generated holds the attribute, the format of the data, the keyword, and the probability of the number of times the agent found the keyword in the data. The lead agent uses both vectors to create the ontology when a second wrapper agent registers in the distributed environment.

The agent also constructs a feature set vector on startup. The feature set vector saves three statistics about the data contained in each column of the database. Specifically the type name, the column size, and decimal digits. The type name stores the type of the data, for example if the attribute name is day and the data is all numbers, the type saved would be ‘integer’. If the data were city names, then the type saved would be ‘character’. The column size statistic saves the maximum length of the column. For example if the type name is ‘integer’, the default column size in Microsoft Access is 4 bytes. If the type were ‘character’, the column size would be the maximum number of characters Microsoft (MS) Access allows, unless the user adjusted the value during the creation of the database. The decimal digits statistic shows whether the data has any fractional digits in its format.

3.2.1.2 Query Agent

The query agent broadcasts its query to the wrapper agent(s) in CoABS. If a wrapper agent has the information the query agent is looking for, the wrapper agent will send the query agent the data. If the wrapper agent does not have the information or does not know if it has the information requested, the wrapper agent checks the ontology mapping to see if the query matches an attribute in the map. If it does, the wrapper agent rebuilds the query with the attribute it is most familiar with, and sends the requested information to the query agent.

3.3 Database Agent Implementation

This research uses Java's JDBC::ODBC Bridge to connect agents to MS Access databases. Once the agent connects to the database, the agent has access to a wealth of information including database attributes, properties of those attributes, and the actual data contained inside the database.

Each agent on start-up calculates metrics for the database it wraps. Each field in the database has metrics generated, and includes an estimate of the field format, the substring similarity vector, the substring frequency vector, and the feature vector. Section 3.3.2 discusses the metric calculations in detail.

Once an agent successfully registers in the environment, the agent can access available information and advertise information contained in their database. As each agent enters CoABS, the agent looks for other agents in the system. Agents carry out all ontology negotiation in the system upon entering the environment. The agents first look for the current ontology 'leader' who is responsible for maintaining the ontology as well as negotiating any changes.

Upon entering an environment with no agents, the agent declares itself leader, sets up an ontology containing only its information, and waits for other agents to enter. If one or more agents are present, the agent that entered last initiates communication with the leader agent, and transmits a subset of its keyword and metric information to the leader. The new agent and the leader will negotiate and make modifications to the global ontology as necessary. Once the lead agent updates the ontology, the lead agent sends the updated ontology to all agents currently in the system. If the leader leaves the environment, and a new agent enters, all the agents currently in the environment elect a new leader. Since every agent has a carbon copy of the global ontology; the system does not rely on any one agent or system maintaining the distributed environment. Figure 3-2 depicts the leader decision algorithm. Section 3.3.1 details the leader election process, Section 3.3.2 discusses the Jaro method, and Section 3.3.3 details the ontology creation process.

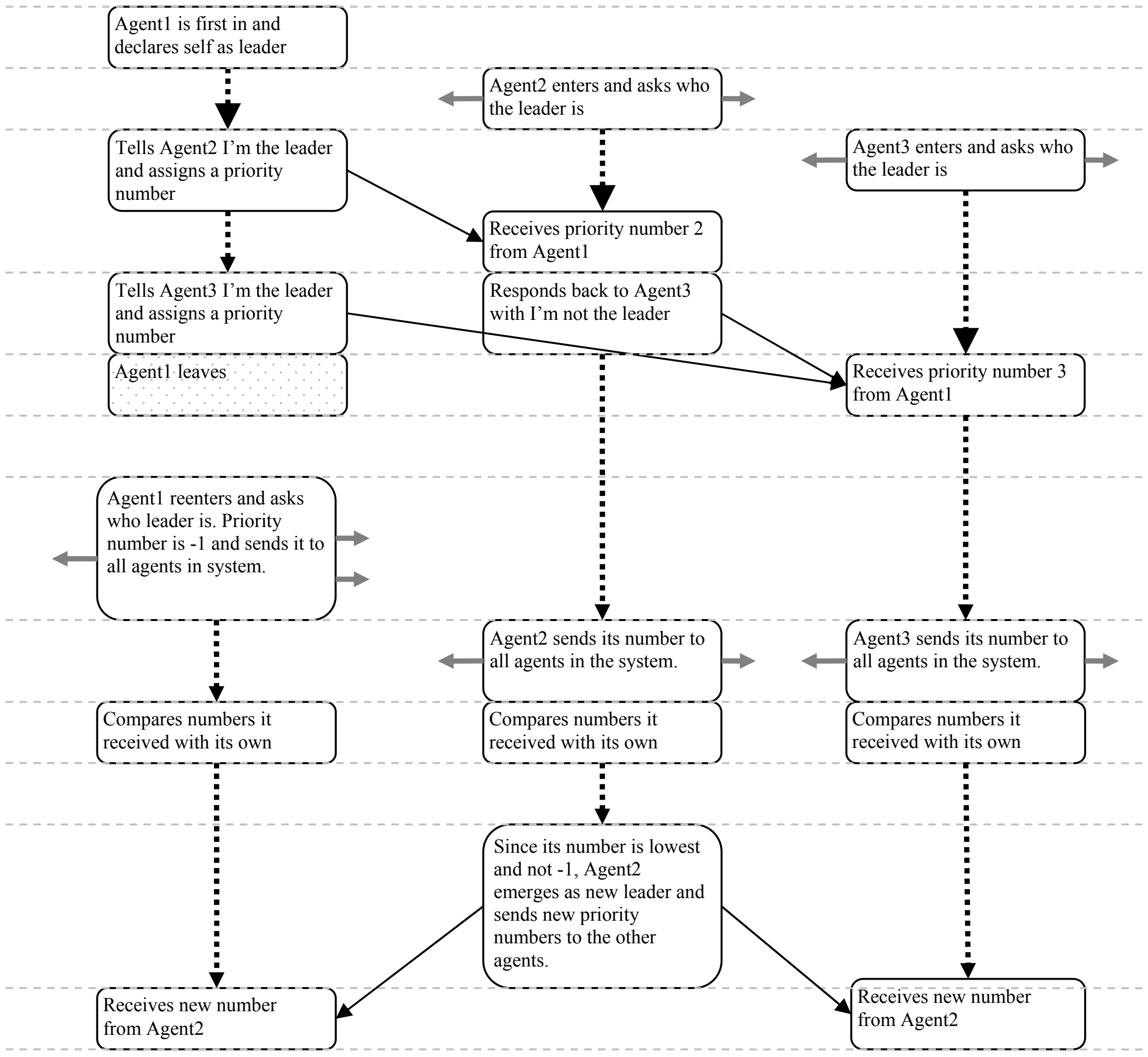

Agent1 is first in and declares self as leader
Agent2 enters and asks who the leader is
Tells Agent2 I'm the leader and assigns a priority number
Agent3 enters and asks who the leader is
Receives priority number 2 from Agent1
Tells Agent3 I'm the leader and assigns a priority number
Responds back to Agent3 with I'm not the leader
Agent1 leaves
Receives priority number 3 from Agent1
Agent1 reenters and asks who leader is. Priority number is -1 and sends it to all agents in system.
Agent2 sends its number to all agents in the system.
Agent3 sends its number to all agents in the system.
Compares numbers it received with its own
Compares numbers it received with its own
Compares numbers it received with its own
Since its number is lowest and not -1, Agent2 emerges as new leader and sends new priority numbers to the other agents.
Receives new number from Agent2
Receives new number from Agent2

Figure 3-2. Agent leader communication

3.3.1 Leader Election Procedure

Figure 3-2 shows how three agents, *Agent1*, *Agent2*, and *Agent3*, select a new leader. For clarity, agent names are italicized when referenced. *Agent1* enters CoABS first and declares itself as leader (priority number 1) because no other agents are present. *Agent2* then enters and searches for other agents. *Agent2* finds *Agent1* and initiates a dialog to determine the leader. *Agent1* responds to *Agent2* and assigns *Agent2* the priority number 2.

Agent3 enters the environment and executes the same transactions that *Agent2* executed. *Agent3* looks for the leader; Agent1 responds and assigns *Agent3* the next priority number in sequence (3).

Agent1 leaves and reenters CoABS. *Agent1* proceeds to inquire *Agent2* and *Agent3* as to the ID of the leader. Since neither is the leader, both *Agent2* and *Agent3* respond to *Agent1* stating that they are not the leader. *Agent1* tracks the replies and when *Agent1* receives all replies from all agents, and *Agent1* does not find a leader, *Agent1* initiates the leader decision procedure.

All agents broadcast their priority numbers. Since *Agent1* just entered the system, *Agent1's* priority number is -1 signifying that *Agent1* does not have a number. Each agent compares the priority numbers. The agent with the lowest priority number and the number is not -1 becomes the new leader and broadcasts to all agents currently in the environment. Finally, the new lead agent (*Agent2*) sends a message to each agent assigning new priority numbers in the same order that each agent contacted the lead agent.

The Jaro similarity metric is the string matching technique used to determine whether two attributes match or not.

3.3.2 Metric Calculation

An estimate of the field format, the substring similarity vector and the substring frequency vector provide the basis for the metrics used to calculate the ontology. Before any agent-to-agent communication takes place, each agent invokes two methods. These methods collect the attribute and sample data from the database that the agent wraps.

The first procedure stores the entire database attributes in a vector. In this research, the database attributes are the column names of the database. For example, the vector shown below in Figure 3-3 is from an educational institution database. The attributes of that database are the contents of the vector. This vector is stored for as long as the agent remains in the CoABS environment.

[Dept, CourseID, Course, HRS, Type, Professor, Day, Time, Bldg, Room, Status]

Figure 3-3. Attribute vector the agent stores

To create a domain ontology, the lead agent compares this vector along with other agent vectors to determine whether an attribute match exists. If it does, the match is stored in the ontology map providing agents with knowledge to answer client queries.

The second method takes samples from the database and stores these in a second vector, the keyword vector. In the event that the lead agent cannot determine a match with just the attribute vectors of the databases, this keyword vector is used to check the actual data contained within the fields to determine if two attributes are similar. If the lead agent determines that the data inside is similar, the lead agent concludes that the attributes are similar and the ontology map is updated.

For this method to work, the method requires the calculation of the keyword vector. This is done by first determining the number of rows (*numRows*) in the database; this is used for two things. First, the agent uses it to calculate the number of samples taken from a database. In this research, the number of rows used for calculating the sample size is bound (α). The α bound used in most examples is 0.20, which makes use of 20% of the samples. The agent randomly selects α * *numRows* from each field and uses this subset to generate the substring vector.

The agent loops through this data subset, and for each two valid strings, the agent compares the two strings to see how similar they are. This comparison uses the Jaro Method [15], discussed in detail in the next section.

If after the Jaro Method the two strings are the same, the agent saves only one string in a vector and a counter is incremented. The counter tracks the number of matches made for each substring element. For example, if there is an attribute called 'instructor' and all of the data inside begins with the string 'Dr.', the agent will process the string through the Jaro similarity metric method and store 'dr.jhn' with the number of times it was found. The agent converts this number into a probability. For example, if the agent found 'dr.jhn' 10 times and the agent pulled 20 pieces of data, the vector will hold 'dr.jhn' and the probability '0.50'. The agent uses this probability when comparing two attributes from different data sources to help determine if the attributes are similar.

After the agent has pulled the data from the first attribute and stored the data with its probability, the agent moves to the next attribute and completes the same process until all attributes are processed. The data structure used for this storage is a doubly linked list with the internal list formatted as:

[attribute name, format, keyword, probability, keyword, probability, …].

The first index of this vector contains the attribute name of the column the agent is working on. The second element of the vector contains the format of the string. The agent uses this format string to help solidify the similarity results between two attributes. The format string represents an expected format of the data, i.e., (###) ###-#### for a 10 digit phone number. The format string calculation attempts to find a general format, representing upper case letters with 'U', lower case letters with 'L', and non-alpha characters with '$'. For example, if one keyword was 'Dr. Doe' the format string would equal 'UL$ULL'; where U is an uppercase letter, L is a lowercase letter, and $ represents a special character; in this example, the $ represents '.', spaces are ignored. When making comparisons, the agent compares the format to determine if there is a format match. To prevent erroneous matches, the format is used as an exact match. This prevents errors such as 'instructor' matching to 'course name'.

In trying to boost result percentages, modifications were made to the format string calculation in the agents. Instead of storing the most general format every time, a process holds the most specific format string. For example, if string1 is 'Dr. John' and string2 is 'Dr. Doe', the format string would save the most specific string1 and compare it with string2. If the characters in the same positions match, the agent saves those characters. If no characters match, one of the symbols U, L, or $ is saved in that characters place. The result is the best format string possible that resembles the majority of the data contained within the database. Back to the example, the format string saved would be ‘Dr. UoLL’. Table 4-10 in Chapter 4 displays the results of this test. The formula for the calculation of the format match is:

$$\forall x, y \qquad \beta = \frac{\sum_{i=1}^{|\Omega|} featureMatch(x_i, y_i)}{|\Omega|} \quad [3.1]$$

where β is the number of feature matches divided by the set of features, Ω. The featureMatch(x, y) variable is a function that compares features and either two features match or they do not. For example, if there is a total of 20 feature set matches possible, and the agent only matched 10 features, then β is 10/20 = 0.5.

The third element of the vector is the Jaro processed keyword, with the fourth element being the probability of the frequency of the keyword. Both the keyword and probability values repeat until the agent processes all of the samples taken for each attribute. As with the attribute vector, this keyword vector is also stored for as long as the agent remains in the environment.

Once the agent leaves CoABS, all feature information is lost. If that same agent comes back in, it must reinitialize to create its attribute and keyword vectors. This keeps the feature setup up-to-date for any changes that occur in the database.

If *Agent1* is the first agent to register in the CoABS environment, *Agent1* invokes the Jaro method to update the ontology map and distribute the map to all agents currently registered in the system. When the second agent registers in CoABS, that agent, who already has its attribute vector and keyword vector, looks for the lead agent. Upon finding the lead agent, the second agent sends that agent its attribute vector and keyword vectors so that the lead agent can update the ontology.

The Jaro method is a method similar to edit distance functions. Distance functions map a pair of strings *s* and *t* to a real number *r*, where a smaller value of *r* indicates greater similarity between *s* and *t* [15]. Similarity functions are the same except that larger values indicate greater similarity [15]. The Jaro metric is not a distance function but does obtain good results. The number and order of the common characters between two strings provides the basis for the Jaro similarity metric [15]. Given two strings *s* and *t*, the Jaro similarity metric is

$$\text{Jaro}(s,t) = \frac{1}{3} * \left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{|s'|}\right) \quad [3.2]$$

where

$|s|$ = Length of string *s*,

$|t|$ = Length of string *t*,

s' = Characters in s which share the same position as they appear in *t*,

t' = Characters in *t* which are common in *s*,

$|s'|$ = Length of string *s'*,

$|t'|$ = Length of string *t'*,

Transposition = Letters in *s'* that do not equal and are not in the same position with the letters in *t'*,

$T_{s',t'}$ = Transposition / 2, and

$$H = \frac{\min(|s|,|t|)}{2}.$$

The Jaro similarity metric is used to compare the attributes and keywords resulting in an automated ontology creation. The lead agent compares the first attribute in its attribute vector with each attribute of the second agent. If the first two attributes match, move on to the next attribute. If the attributes do not match, the agent sends those two attributes to the Jaro method.

In step one of the Jaro method, the lengths of each attribute is stored in integer variables. *H* is calculated using these lengths.

In the next step, character arrays store attribute strings so the agent can examine and test the characters more easily. In addition, a conversion of the attributes to lowercase letters takes place as they enter the character array.

In keeping with the notation in the formula, let *s* equal the lead agent's attribute and let *t* equal the second agent's attribute. The next step is to determine the letters in *s* that are common with *t* and the letters in *t* that are common with *s*. *H* is used to calculate a 'moving' window. A lower bound and an upper bound define this window. Upon reaching the upper bound, the window moves. Once reaching the upper bound, the agent calculates a new lower and upper bound, the agent makes character comparisons, and the window moves again. This process continues until reaching the ends of the character arrays. For example, let *s* = 'Help' and *t* = 'HelpMe'. In the first step, the agent converts both strings to lowercase letters and puts them into character arrays, Figure 3-4.

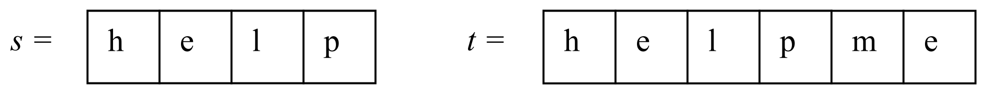


Figure 3-4. Example Character Arrays

The agent calculates the lower and upper bounds with a loop index and H. Referring to Figure 3-4 and comparing *s* with *t* the following for loop applies:

```
for (int i=0; i < s.length; i++)
    The lower bound ← i – H ← 0 – 2 ← -2, when i ← 0.
    The upper bound ← i + H ← 0 + 2 ← 2, when i ← 0.
```

Since the lower bound falls outside the dimensions of the array that bound is set to zero if it was the lower bound or its set to the array length if the upper bound value went outside the dimension of the array. For the first iteration described above, the window is bounded from element zero to element 2, which correspond to the letters 'h', 'e', and 'l' in the *t* character array. This method takes the first letter of *s* and compares it to all letters in the window. The first letter in array *s* is 'h'. This matches the 'h' in the *t* window so the character 'h' is saved in the string *s'* and an '*' replaces the 'h' in the window to show that the element has been visited. Once 'h' is compared with the letters in the window, the index is incremented, new lower and upper bounds are calculated, and the string *s'* is built. This same process is used to build the *t'* string and the window moves through the *s* character array instead of the *t* character array.

So far in the Jaro method, |*s*|, |*t*|, *s'*, and *t'* are calculated. The next step is to calculate the transposition. This is a straightforward computation and works as follows: the agent compares *s'* and *t'* letter-by-letter. If the letters do not match, a counter is incremented.

After comparing the two prime strings *s'* and *t'*, the counter is divided by two and this result is the transposition number.

The agent compares both *s'* and *t'* against its keyword vector. If the agent finds either of these words in its vector, that keyword's number quantity is incremented. If the agent does not find either *s'* or *t'* in its keyword vector, the agent adds these words to the keyword vector with a number quantity of 1 for each, because it was the first time the agent found these words. The agent does not calculate the Jaro number for the substring vector because the calculated number is not used here.

Finally, the *s'* and *t'* string lengths are found and the Jaro similarity metric is computed for these two attributes. Once the Jaro number is calculated the Jaro number, *s*, and *t* are sent to the method updateOntology() to build and update the ontology.

3.3.3 Building the Ontology

When a new agent enters the environment and transmits its metrics to the leader, the first thing the leader does is compare each field in its field vector with the fields of the newly entered agent. If the field names of two attributes match, move on to the next attribute. If the attributes do not match, the similarity of the two fields is calculated.

For calculating two fields' similarity, the agent sends each keyword through the Jaro method to obtain a Jaro number. Remember, along with the keyword the probability number and the format match is stored. For the set of best substring keyword matches, the agent multiplies the substring probabilities together with the Jaro number, and then sum over all matches. Once all of the keywords for these attributes are processed, the agent analyzes the summation of the probabilities and Jaro numbers. If the summation is above confidence threshold (τ) and the format match is the same, the field match is confirmed and the ontology is updated. Otherwise, the agent discards the two attributes and two new attributes are processed. The formula for the calculation of an attribute match is $\sum_{keyword1, keyword2} \Pr(keyword1) * \Pr(keyword2) * jaroNum * \beta$ [3.3]

where the confidence threshold (τ) determines how similar a match is based upon the metric comparison information and the feature match value. For all testing in this research, τ is set to 0.80, meaning that the agent is 80% confident the two words its comparing are similar. The higher the metric comparison, the more confidence there is a match. If the confidence threshold is set low, some of the results will likely be erroneous. There is a delicate balance in setting the threshold as high as possible while still obtaining all of the attribute matches the agents should find. Table 3-1 shows a sample-negotiated ontology.

Table 3-1. Negotiated Ontology between agents in CoABS

Automated Agent Ontology

ACFT_QTY	Aircraft_QTY
WPN_QTY	WPN_Quantity
ACFT_TYPE	ACFT_TYPE
WPN_Quantity	WEAPoN_QTY
Aircraft_QTY	Aircraft_QTY
WPN_Name	WPN_Name
PROB_DAMAGE	PROB_DAMAGE_ TOTAL
WEAPoN_QTY	WPN_QTY
PROB_DAMAGE_ TOTAL	DAMAGE
DAMAGE	PROB_DAMAGE

Table 3-1 is the ontology created that every agent in the environment stores. One can see which attributes the agents matched. For example, agents concluded that WPN_QTY is the same or similar to WPN_Quantity. When a client queries for WPN_QTY, the agent first looks at its attribute vector. If the agent has WPN_QTY as one of its attributes, the agent returns the information. If the agent does not have WPN_QTY as one of its attributes, the agent will look to this table and see if there is a match with its attributes. If a match exists, the agent will rebuild the query with the new value and return the information to the querying client. The last step is to query the wrapper agents with a querying agent to test the accuracy of the ontology.

3.4 Query Process

A query agent makes queries to other agents in the CoABS system. It must register with the environment just like the database agents. After the query agent successfully registers, the query agent searches for wrapper agents in the environment. The query agent targets this search either to specific agents or to all agents, depending upon what the client needs information on. For example, if the query agent wants to query *Agent1* for course names, instead of getting a list of all course names from all the agents in the system, the query agent will send a message specifically to *Agent1*. When the query agent finds the agent or agents it is looking for, the query agent sends a SQL query statement to those agents in CoABS.

After the query agent sends the message, the receiving agents process the message. The wrapper agents compare the SQL string with their individual attribute vectors. If the words are in the wrapper agent's attribute vector, the agent retrieves the requested information and sends it back to the querying agent. If the wrapper agent does not recognize an attribute in the query, the agent will check the ontology map for the word. If the agent finds the word in the ontology map, the agent rebuilds the query with the new word, and the agent retrieves the information and sends it back to the querying agent. If the agent does not find the word in the ontology map, the wrapper agent does not reply to the query because it does not have the requested information.

For example, if the query received is SELECT ACFT_QTY FROM Weapon, *Agent1* will process each word of the query. The first word SELECT is a SQL keyword, so *Agent1* ignores this word and grabs the next word, 'ACFT_QTY'. If 'ACFT_QTY' is not in *Agent1's* attribute vector, *Agent1* checks its ontology map. If *Agent1* finds 'ACFT_QTY', the map returns its equivalent. In this example, the equivalent to 'ACFT_QTY' is 'Aircraft_QTY', Table 3-1. *Agent1* replaces the word 'ACFT_QTY' with 'Aircraft_QTY' and finishes processing the SQL string which now looks like SELECT Aircraft_QTY FROM Weapon. *Agent1* has 'Aircraft_QTY' in its attribute vector; therefore, *Agent1* retrieves the information and sends it back to the querying agent.

3.5 Summary

This chapter discusses the goals and requirements for successfully automating ontology creation in the JBI. It also summarizes the approach used to satisfy the requirements. This chapter also discusses the implementation details of this research. This system, as detailed in this chapter, provides the initial concept for automated ontology creation on large databases within the JBI. Although not perfect, this system provides the ability to relax some formatting restrictions in the JBI environment. For example, clients will no longer have to provide exact spelling on attributes. The syntax can be close, and the agents will recognize that “windspeed” is the same as “Windspeed”. Chapter 4 discusses the testing environment and the results of those tests.

4. Analysis and Results

This chapter discusses the evaluation of the domain ontology generating agents wrapped around similar databases and their performance when creating an ontology as defined in Chapter 3. This chapter also measures the effectiveness of leader negotiation as implemented from Chapter 3. The testing used thirteen queries to measure agent ontology mapping accuracy. To achieve 100% accuracy, all agents must return correct responses to the queries. With the exception of the five agent test, each set of queries is run on each different combination of the order the agents enter the CoABS environment. The combination queries run under two conditions, before and after a leader negotiation.

All agents exist independently. Each agent can function and process queries without other agents in the system. Communication during leader negotiation satisfies the agent autonomy requirement by providing agent communication and the means to negotiate who is in charge. When the lead agent leaves the environment and another agent enters, leader negotiation takes place. Only one agent will emerge as the new leader, and ontology negotiation will commence thereafter. Section 4.2 shows the result of both agent ontology creation and the leader negotiation.

4.1 System Evaluation

The entire agent architecture allows a user to interact with it as if it were one large database. When query agents register in CoABS to make standard SQL queries to access data from the distributed environment, the client does not need to know which database or which wrapper agent to query. The client simply makes the query and the wrapper agents return the information whether it is from one agent or from all of the agents currently registered in the environment.

4.1.1 Ontology Creation

As each agent enters the CoABS environment, the agent assesses its surroundings and determines how to proceed next in creating the ontology. If it is the only agent in the environment, there is no agent negotiation for an ontology. When the second agent enters the environment, this second agent recognizes the first agent as the lead agent and proceeds to initiate communication and negotiation. The result is an agent created ontology that is used by the agents when query agents access the system.

During a query, the wrapper agents look at their data. If they do not have the requested attribute name, the wrapper agent will look to the ontology map for a translation. If there is a translation in the mapping that references one of its attributes, the wrapper agent translates the query and sends the requested information back to the querying agent. If there is no translation in the ontology, the wrapper agent simply does not reply to the query. Using this technique prevents unnecessary communication between agents.

4.2 Three Agent Results

For the 3 agent results, three wrapper agents wrap databases that contain data split from a simulated JBI information resource. The databases contain data pertaining to aircraft mission sorties, including targeting information and the attributes as follows: Aircraft quantity (ACFT_QTY), Aircraft Type (ACFT_TYPE), Date and time created (DATETIME_CREATED), Date and time last changed (DATETIME_LAST_CHG), Probable damage total (PROB_DAMAGE_TOTAL), Weapon name (WPN_NAME), and Weapon quantity (WPN_QTY). Changes made to each database attribute simulate different columns that contain similar data. Appendix A shows each agent with its corresponding database attributes and how they compare.

To simulate erroneous data entry, incomplete data, and statistically different data, modifications are made to the data in each database. Data is modified: sometimes a field is deleted, a number changed, or the data for the particular field made completely unrecognizable. These results show two aspects of this research. The first is the success rate of the automated ontology creation, the second is how well the wrapper agents decide who the next leader is, and the impact that has on the ontology.

This section compares the agent created ontology and an expert created ontology to evaluate how well the agents created the ontology. Table 4-1 shows the expert ontology mapping.

Table 4-1. Human Expert Ontology

ACFT_QTY	ACFT_Quantity
Aircraft_QTY	ACFT_QTY
ACFT_Quantity	Aircraft_QTY
ACFT_TYPE	Aircraft_TYPE
DATETIME_CREATED	DATETIME_CREATED
DATETIME_LAST_CHG	DATETIME_LAST_CHG
PROB_DAMAGE_TOTAL	DAMAGE
PROB_DAMAGE	PROB_DAMAGE_TOTAL
DAMAGE	PROB_DAMAGE
WPN_NAME	Weapon_NAME
WeaPoN_NAME	WPN_NAME
Weapon_NAME	WeaPoN_NAME
WPN_QTY	WPN_Quantity

Table 4-2 shows the agent ontology mapping, which performed the best, before leader negotiation. This mapping produced a 62% agent response with *Agent3*, *Agent2*, then *Agent1* entering the system in that order, and with each agent generating substring statistics (α) for 80% of the records, but only transmitting the best 20, with the threshold (τ) set at 80%.

Table 4-2. Agent Ontology Matching Before Leader Negotiation

Weapon_NAME	WPN_NAME
ACFT_QTY	DATETIME_CREATED
DATETIME_LAST_CHG	DATETIME_LAST_CHG
ACFT_TYPE	ACFT_TYPE
WPN_NAME	WeaPoN_NAME
PROB_DAMAGE_TOTAL	PROB_DAMAGE
DATETIME_CREATED	DATETIME_CREATED

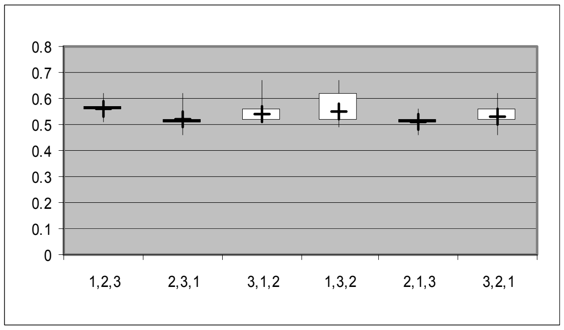
As can be seen when comparing Table 4-1 with Table 4-2, the agent generated ontology matched the string based fields properly, ACFT_TYPE and WPN_NAME. However, the numeric fields, the ontology tended to mismatch, ACFT_QTY, DATETIME_LAST_CHG. This is due to the matching method being predominantly string based. With the numerical methods, combining the substring match with the information used by SemInt better results should be possible.

Periodically, after the agents renegotiate a new leader, the ontology created contains duplicate and erroneous mappings. The reason is agent ontology creation is dynamic. The lead agent updates and distributes a new ontology every time a new agent enters the environment. After a leader negotiation, the agent that requested a leader sends its database vectors to the new leader making an ontology creation start again. Table 4-3 shows the best mapping created by the agents that obtained a 79% agent response rate to the 13 queries used in testing, with an ontology very similar to the expert generated version in Table 4-1.

Table 4-3. Agent Ontology Matching After Leader Negotiation

Weapon_NAME	WPN_NAME
ACFT_QTY	ACFT_QTY
WPN_QTY	WPN_QTY
DATETIME_LAST_CHG	DATETIME_LAST_CHG
ACFT_TYPE	ACFT_TYPE
Aircraft_QTY	Aircraft_QTY
WeaPoN_NAME	WPN_NAME
WPN_NAME	WPN_NAME
PROB_DAMAGE	PROB_DAMAGE_TOTAL
PROB_DAMAGE_TOTAL	PROB_DAMAGE_TOTAL
DATETIME_CREATED	DATETIME_CREATED
DAMAGE	PROB_DAMAGE_TOTAL

Figures 4-1, 4-2, 4-3, 4-4, and 4-5 below show how the agents responded to queries after negotiating the ontology using varying amounts of the database to build the substring vector. The bound (α) is the value of the sample size taken from the database. The α bound used for creating the sample sizes in generating the substring vectors were 20%, 40%, 60%, 80%, and 100%, respectively, with only the best 20 substrings being transmitted and used for ontology creation. The 100% data sample only uses each piece of data and is not an exhaustive n*(n-1) matching of data items. The confidence threshold (τ) is set at 0.80. The graphs show how the agents responded correctly to a query over the 13 queries in six different runs. The results in Figures 4-1 and 4-2 shows that the agents responded correctly over 50% of the time and that the more data used for substring creation, the better the generated ontology.



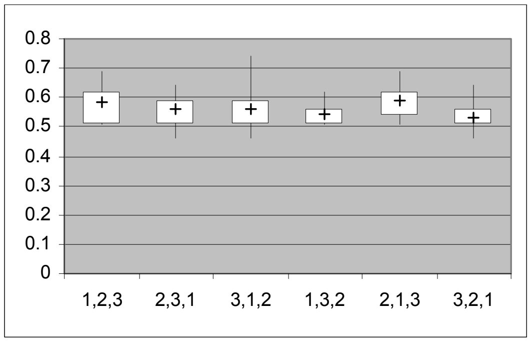
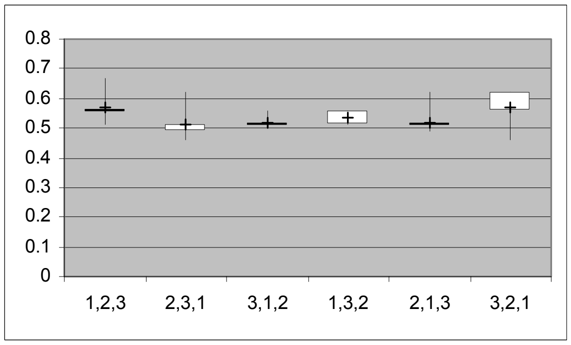


Figure 4-1. 20% Query results before (left) and after (right) leader negotiation



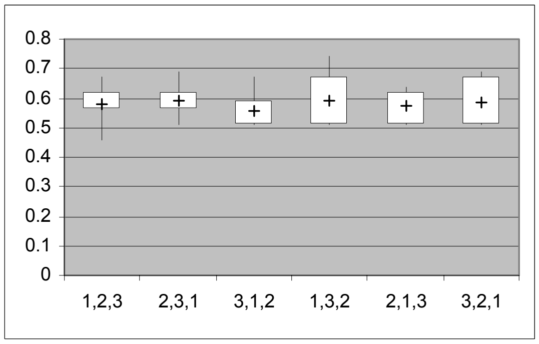
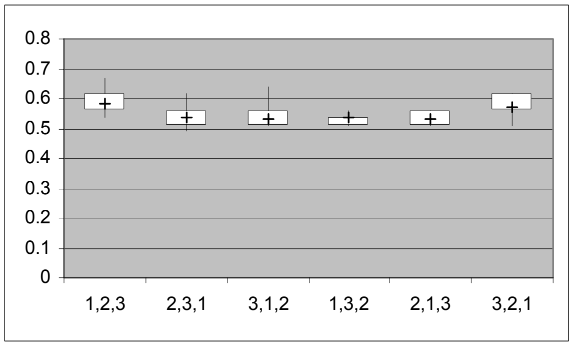


Figure 4-2. 40% Query results before (left) and after (right) leader negotiation



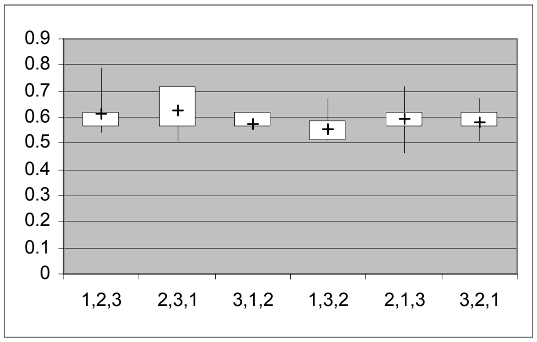


Figure 4-3. 60% Query results before (left) and after (right) leader negotiation

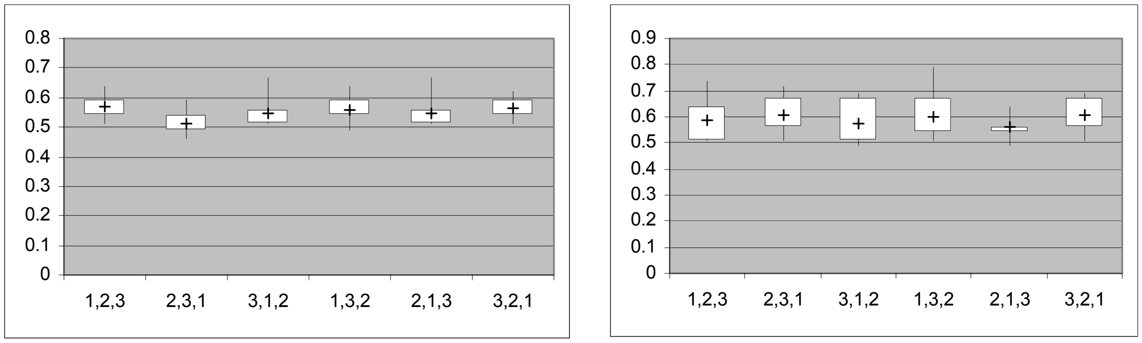


Figure 4-4. 80% Query results before (left) and after (right) leader negotiation

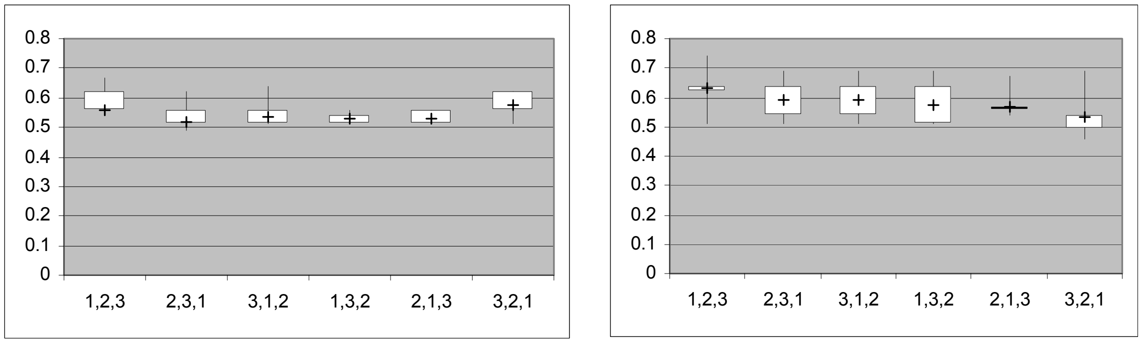


Figure 4-5. 100% Query results before (left) and after (right) leader negotiation

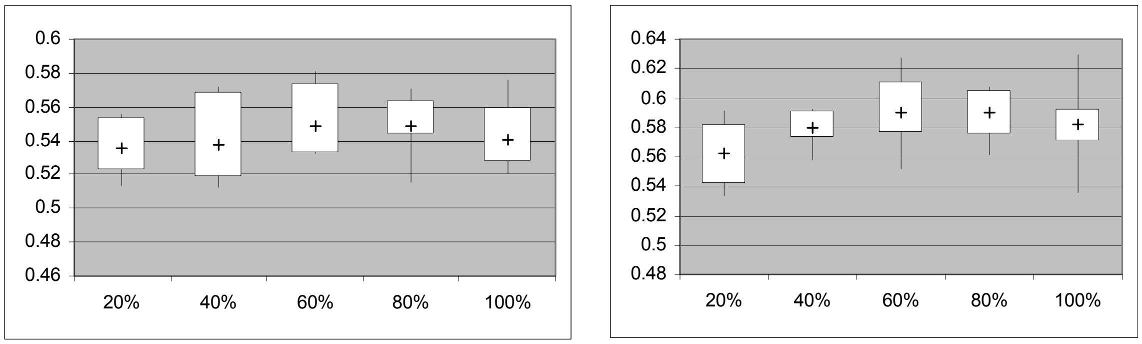


Figure 4-6. Summary query results before (left) and after (right) leader negotiation

Figure 4-6 displays the average summary results before and after the leader negotiation. The percentages along the x-axis of the graphs are the α bound percentages pulled from the databases. These percentages map to the percentages in the tables in

Figures 4-1 through 4-5. All results show that the agents responded correctly at least 51% of the time. These two tables also show that after the leader negotiation, the generated ontology is better. The reason for this is that instead of being a statistical combination of all of the agents' information, agents build the ontology using a comparison between the leader and the new agent.

Based on the before leader negotiation Figures 4-1 through 4-5, the ontology creation accuracy falls between 50% and 61% on average. It is easy to see that no matter what order the agents enter CoABS the average results are statistically the same. The median values before leader negotiation in Figure 4-6 shows that the mean falls between 53% and 55%. Just as with the agent order, no matter how much data is sampled from the database the ontology creation accuracy is statistically the same. This is significant because if communication bandwidth requirements are an issue, the agents only need to transfer 20% of the data to negotiate an ontology and achieve these results. There is no need to sample the entire database when the result is going to be the same.

Analyzing the after leader negotiation results in Figures 4-1 through 4-5, the results are similar but the percentages are a little higher. The accuracy falls between 51% and 71%. The results are better because when agents negotiate a leader, first, the agents already have an ontology from the first negotiation, and second, the agents have a second chance of correcting and making correct mappings adding to the already created ontology. Therefore, the results are slightly higher and the mean values in Figure 4-6 also support this conclusion.

One reason for the errors in the agent created ontology resulting in the query errors is when agents build their keyword vectors, the agent uses sample percentages of the data in the database to build the keyword vectors. With the exception of the 100% test, since the sample percentage is stochastic, every time an agent negotiation takes place, the results are slightly different from the previous run. This is why on certain runs PROB_DAMAGE mapped to WPN_NAME. It is possible that on a subsequent run of the agent negotiation, the agents will not produce the PROB_DAMAGE to WPN_NAME mapping.

The addition of a feature set vector and a new format match method improves the algorithm by achieving slightly better results. Chapter 3 explains these improvements in detail. The improved code ran the experiment set up that produced the best result from the original code Figure 4-7 shows that the improved code performed no less than 56% and no higher than 72%, whereas the unimproved code peaked low at 51% and high at 79%. As before, the numbers along the x-axis show the order in which the agents enter the environment.

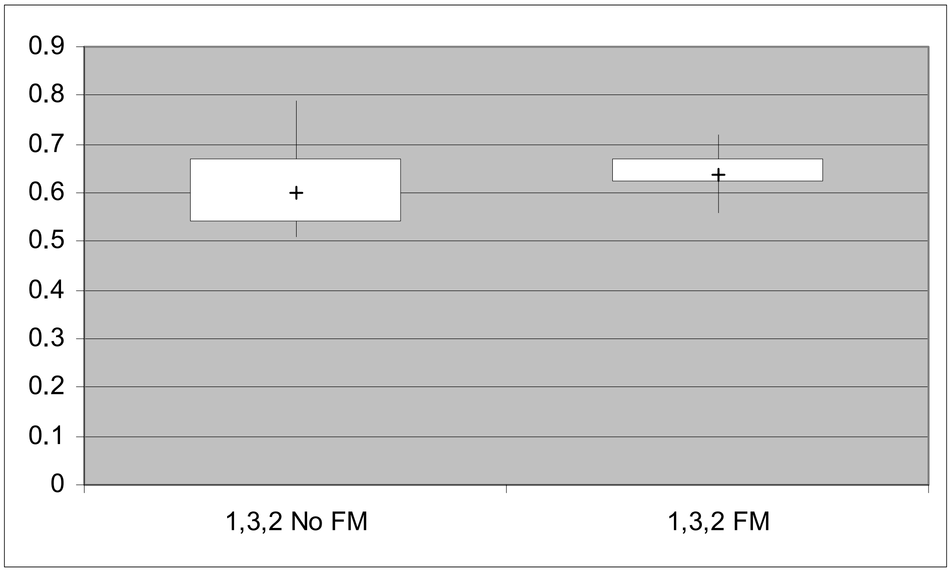


Figure 4-7. 80% Query Result without Feature Match Improvement (left) and with Feature Match Improvement (right)

To see how ontology creation and leader negotiation performs with more than three agents, two more agents with different databases where added to the environment. The database attributes for *Agent4* and *Agent5* are in Appendix A.

Table 4-4 shows the proper mapping that should take place between all five agents. After 10 runs, using an α of 0.80 sample size, Tables 4-5 and 4-6 show the best ontology created between the five agents.

Table 4-4. Human Expert Ontology Matching with five Agents

ACFT_QTY	ACFT_Quantity
Aircraft_QTY	ACFT_QTY
ACFT_Quantity	Aircraft_QTY
ACFT_TYPE	Aircraft_TYPE
DATETIME_CREATED	DATETIME_CREATED
DATETIME_LAST_CHG	DATETIME_LAST_CHG
PROB_DAMAGE_TOTAL	DAMAGE
PROB_DAMAGE	PROB_DAMAGE_TOTAL
DAMAGE	PROB_DAMAGE
WPN_NAME	Weapon_NAME
WeaPoN_NAME	WPN_NAME
Weapon_NAME	WeaPoN_NAME
WPN_QTY	WPN_Quantity
CATEGORY	CAT
COORD_LT	COORD_LAT
COORD_LONG	COORD_L
COORD_L_ORD	COORD_LONG_ORD
DATETIME_CREATED	DATE_CREATED
DATETIME_LAST_CHG	TIME_LAST_CHG

Table 4-5. Best Agent Ontology Creation with Five Agents 1

COORD_LONG	PROB_DAMAGE
WPN_QTY	WPN_Quantity
COORD_L	PROB_DAMAGE
Aircraft_TYPE	PROB_DAMAGE
WPN_NAME	PROB_DAMAGE
WeaPoN_NAME	Weapon_NAME
ACFT_TYPE	WPN_Quantity
Aircraft_QTY	DMPI_ID
CAT	PROB_DAMAGE
ACFT_Quantity	WPN_Quantity
DATE_CREATED	Aircraft_QTY
TIME_LAST_CHG	Aircraft_QTY
ACFT_QTY	Aircraft_QTY
PROB_DAMAGE_TOTAL	PROB_DAMAGE
DMPI_ID	Aircraft_QTY
DAMAGE	PROB_DAMAGE

Table 4-6. Best Agent Ontology Creation with Five Agents 2

DATETIME_LAST_CHG	DATETIME_LAST_CHG
FAC_NAME	FAC_NAME
NO_STRIKE	NO_STRIKE
COLLATERAL_DAMAGE	COLLATERAL_DAMAGE
FUNCT_PRIMARY	FUNCT_PRIMARY
COORD_LT	COORD_LT
COORD_LONG_ORD	COORD_LONG_ORD
WPN_Quantity	WPN_Quantity
TGT_OBJ_NAME	TGT_OBJ_NAME
OPER_STATUS	OPER_STATUS
COORD_L_ORD	COORD_L_ORD
EVAL	EVAL
MSN_TYPE	MSN_TYPE
PROB_DAMAGE	PROB_DAMAGE
DATETIME_CREATED	DATETIME_CREATED
CONDITION	CONDITION
CC	CC
TGT_DTL_NAME	TGT_DTL_NAME
COORD_DERIV	COORD_DERIV
COORD_LAT	COORD_LAT
COORD_ROA	COORD_ROA
COORD_LAT_ORD	COORD_LAT_ORD
REMARK	REMARK

4.3 Five Agent Results

Using the same settings as the three agent results, negotiation is tested with five agents. This test determines if an increase in the number of agents would have an effect on the applicability of the generated domain ontology.

Table 4-5 shows an ontology result with agents entering the environment in 1, 2, 3, 4, 5 order, pulling 80% of their database data, $\alpha = 0.80$. The agents only correctly matched 29% of the possible attribute matches. Table 4-6 lists the mappings that are common in two or more databases. Table 4-4 shows the mappings the five agents should have negotiated, 17 in all. Figure 4-8 shows the ontology creation results with five agents. The data was taken from 10 runs of the five different α bounds.

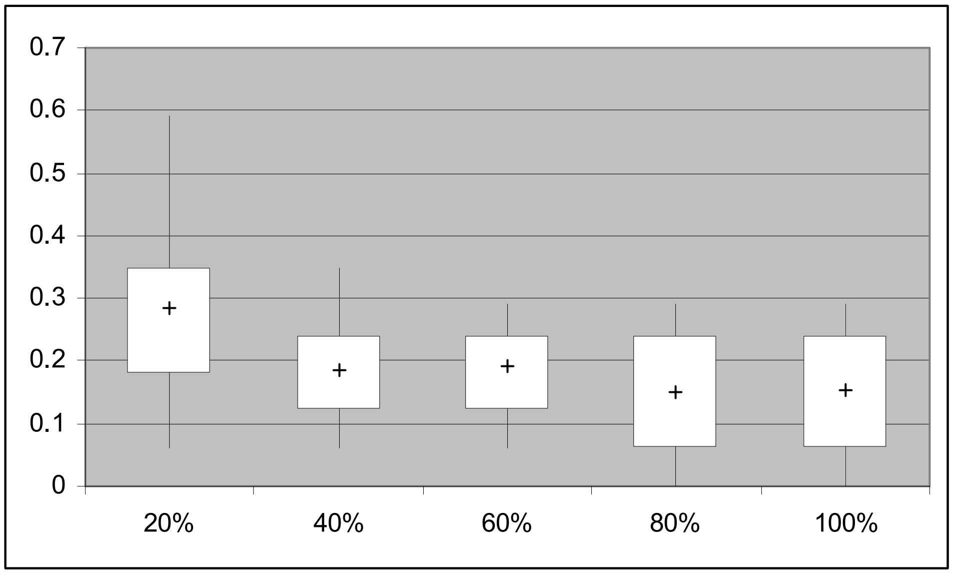


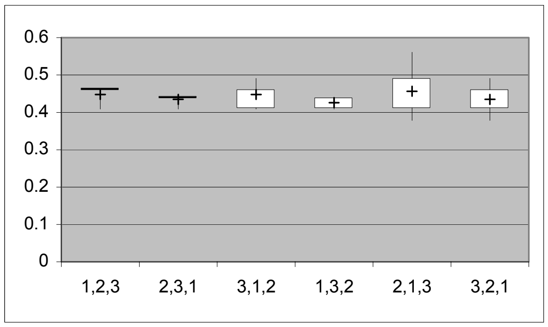
Figure 4-8. Five Agent Ontology Results

Figure 4-8 shows agents created the best ontology when α is 0.20. Both the 0.80 bound and the 1.0 bound data resulted in ontology creation where none of the 17 required matches matched. This indicates that as more agents enter the system, the more difficult the ontology generation becomes. A way to alleviate this is to cooperatively create the ontology by eliminating the sole leader. Essentially, each agent has the ability to create, modify, and finalize an ontology. However, this alternative is more memory and communication intensive because each agent must maintain a representation of other agents' data.

4.4 Large Dataset Results

Testing was accomplished with the sample size (α) set at the five different settings 0.20, 0.40, 0.60, 0.80, and 1.00, the confidence threshold (τ) is set at 0.80, and only the best 20 substrings are transmitted to the lead agent for ontology negotiation.

This database is larger than the previous with 6 attributes and over 4700 rows of information. Appendix B shows the agents used and their attribute values. Appendix B also shows the proper mappings that should take place for a successful agent ontology negotiation. Figures 4-9 through 4-13 show the results of those tests. Figure 4-14 shows the summary results of all percentage queries before and after leader negotiation using the same α bounds as in the previous tests.



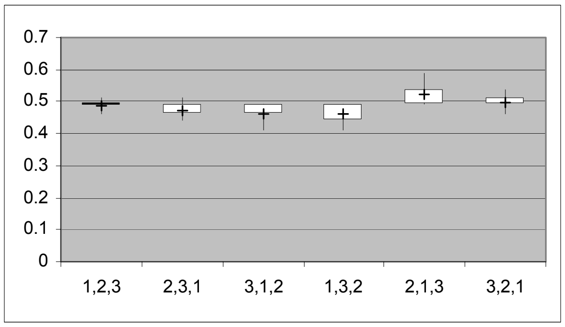
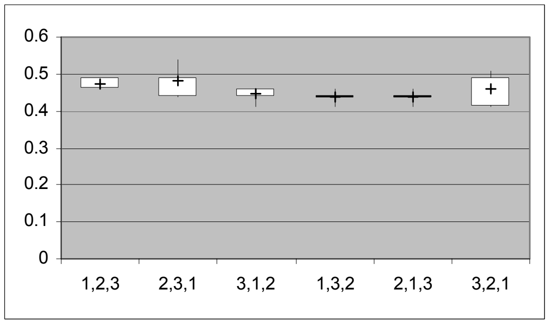


Figure 4-9. 20% Oracle data query results before (left) and after (right) leader negotiation



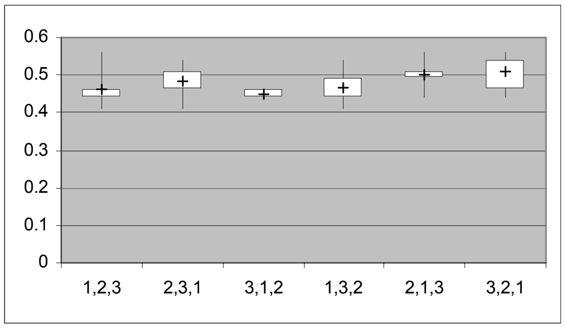
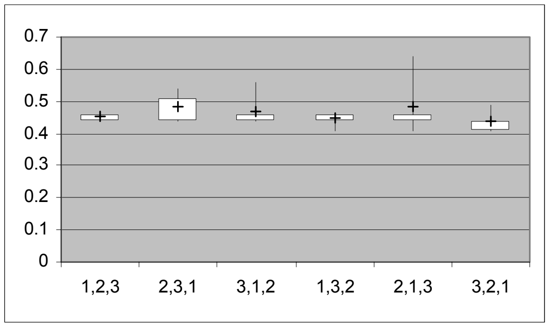


Figure 4-10. 40% Oracle data query results before (left) and after (right) leader negotiation



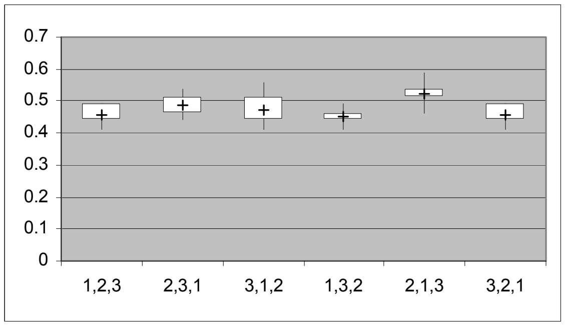
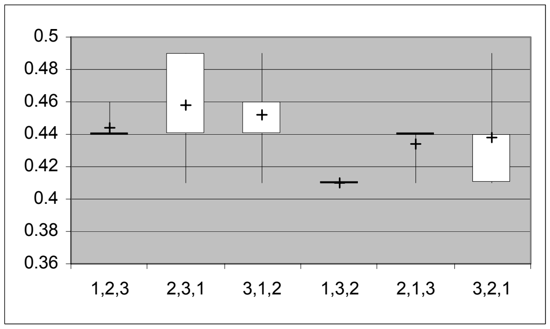


Figure 4-11. 60% Oracle data query results before (left) and after (right) leader negotiation



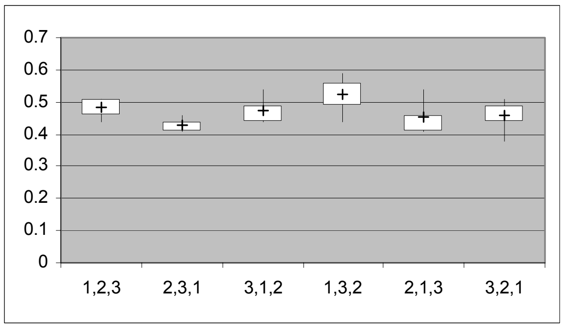
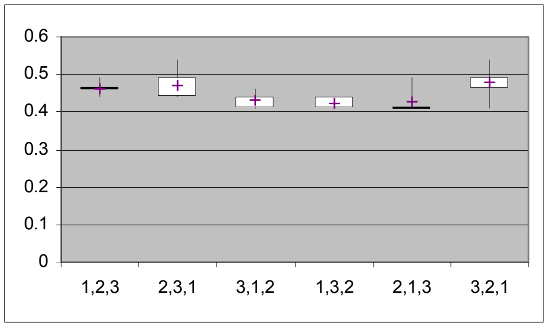


Figure 4-12. 80% Oracle data query results before (left) and after (right) leader negotiation



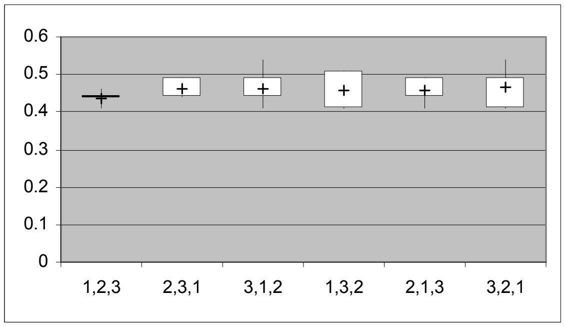
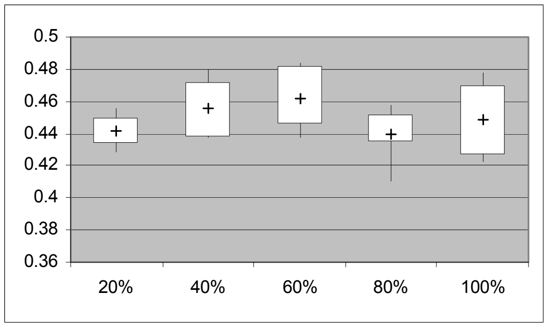


Figure 4-13. 100% Oracle data query results before (left) and after (right) leader negotiation



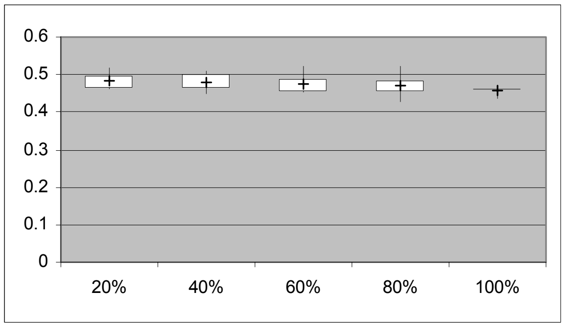


Figure 4-14. Oracle summary query results before (left) and after (right) leader negotiation

This set of tests ran 13 queries against three agents. As in the previous tests, there are a possible 39 responses and the best result achieved 64% match success rate, with an α of 0.60 before leader negotiation, and the agents entering the environment in *Agent2*, *Agent1*, and *Agent3* order. Figure 4-11 shows this result. This ontology creation was more difficult for the agents that the previous. Two of the six attribute fields are number fields and the agents have a difficult time determining whether the attribute match exists or not. These fields are mapped based upon the keyword data making the resulting queries incorrect.

The box and whisker plots in Figures 4-9 through 4-13 are comparable to the smaller dataset results above. The before leader negotiation results support the conclusion that even with a large dataset, no matter how the agents enter CoABS, the accuracy of the ontology created is statistically the same. The results fall between 41% and 50%. After leader negotiation, results show less variance with the large dataset because just as with the small dataset results, the agents have a second chance to correct mappings in the ontology making the results slightly better. The results fall between 42% and 52%. The median summaries in Figure 4-14 also support this conclusion.

4.5 String Only Results

This result shows how this research technique performs when the databases contain only strings, no numbers. For this test, three agents were used and the databases contain 7 attributes and 306 rows of information. Appendix C lists the attribute values for the three agents used. The agent settings are the same as in the previous tests, the sample size (α) is set at 0.80, the confidence threshold (τ) is set at 0.80, and only the best 20 substrings are transmitted to the lead agent for ontology negotiation. Figure 4-15 shows this result.

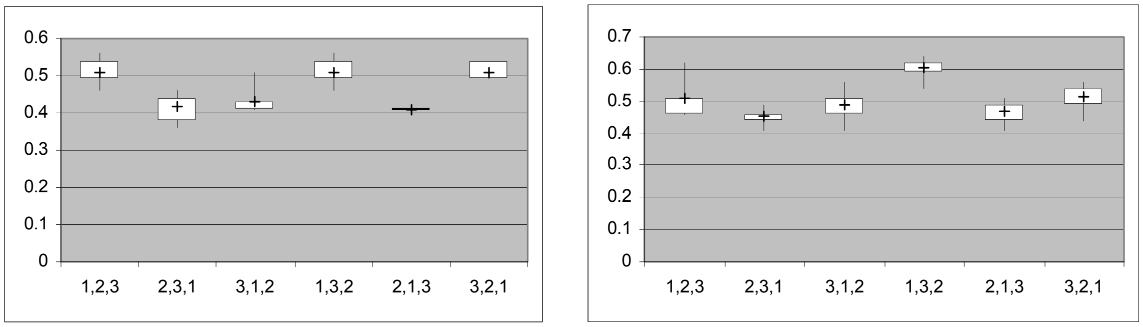


Figure 4-15. 80% String only query results before (left) and after (right) leader negotiation

The string only best result was 64% match success rate after leader negotiation with the agents entering in *Agent1*, *Agent3*, and *Agent2* order. These results are comparable to the previous testing, but more testing is required to see if there is more improvement when dealing strictly with strings, or if it makes no difference what kind of data is being processed.

One last test was accomplished to test the theory that if all databases in the test were the same, except for the attributes, then the Jaro similarity metric should perform better than the string only case. This test pulled 20% sample data and ran 13 queries. There is a possible 39 correct responses as in the previous tests. The match accuracy result was only 36%. This leads to one of two conclusions. Either 79% is the best that the Jaro Similarity Metric can do with mixed data or due to the sampling of the data that generates the keyword vector, the process is stochastic not deterministic. The result is that this method will most likely not achieve a true 100% match without additional information. However, a 79% match by making a comparison with the best 20 keywords from a 20% sampling of the data is very good.

4.6 Research Benefits

The benefits of this research are two-fold. It provides a feasible way to automate agent ontology creation and each agent involved is autonomous.

Results show that an automated ontology creation is possible relieving the human expert from manual mapping the data. Other research experienced comparable results, but with some using external techniques in the matching. For example, the best matching SemInt achieved was 44% for 2578 attributes in 293 tables, with 1760 data elements [22].

Using string matching algorithms and techniques without external data dictionaries or thesauruses, the agents in the environment compare words and come up with a similarity metric. Depending on the value of the metric, the agents decide whether the words are similar and the results are stored for all agents to use. Since each agent has a copy of the ontology data retrieval is quick. If another agent entering CoABS updates the ontology, the agent sends the new ontology map out to all agents again ensuring global consistency throughout the environment.

This research also keeps all agents autonomous. Every wrapper agent has the same code as every other agent. This means that if the lead agent leaves the environment all other agents can still function and process queries. Only upon a new agent entering CoABS does ontology negotiation take place. In this case, the agents decide amongst themselves which agent will emerge as leader and provide the ontology negotiation facilities to the new agent updating the ontology as needed.

4.7 Summary

These results show that automated agent ontology creation is feasible. The three agent experiments produced results no less than 51% correctness while other research efforts produced results anywhere from less than 20% to over 98% correctness. The five agent and string only experiments produced results no less than 39%. In order for these other research efforts to achieve such high correctness percentages, they use data dictionaries, thesauruses, neural networks, and some used combinations of techniques together. This research uses no outside help and works solely on the attributes, data, and metadata information. Implementing the two code enhancements, specific to general format matching and feature set vector, improved the results slightly. Chapter 5 discusses the conclusion and suggests ways to expand this research.

5. Conclusions and Recommendations

This research provides a multi-agent methodology and implementation that provides services for accessing multiple information sources, each of which make use of different data and message formats. Currently an expert integrates these data sources by hand. The implementation of this research proposes automatic mapping of the relationships using intelligent agents. These intelligent agents communicate and negotiate an ontology thereby eliminating the need for an expert to develop the ontology by hand. The ontology created is as dynamic as the agents themselves are. When a new agent enters the distributed environment, the lead agent negotiates and distributes a new ontology to all agents in the environment.

Coupled with this approach all the agents are autonomous and can negotiate, update, and distribute the ontology. When the agents enter the environment, a leader is decided and that leader is the one that commands all ontology negotiations until that agent leaves the system.

This solution, automated agent ontology creation, meets the criteria for a successful implementation to automated ontology creation. The wrapper agents enter the CoABS environment, transmit vectors to the lead agent, and the lead agent invokes the Jaro similarity metric method. The lead agent updates and distributes the newly created ontology to all agents in the environment. In comparison to related work, this solution proved to be more simplistic and require little or no human expert for manually mapping an ontology to integrate distributed database systems. In satisfying the criteria for a successful implementation, this research lays the groundwork for a solution to automate agent ontology creation.

This solution is feasible for the DOD to implement. Tackling the interoperability problem, this research enables a JBI implementation that does not need the XML exact match implementation. In addition, experts need not fully scrub and correct data in their databases before publishing their information to the JBI.

5.1 Recommendations for Future Research

This research is a step toward automating ontology creation. Using this method in the JBI will enable some JBI restrictions to relax and make it a more viable solution to implement. This research requires more work to ensure that the ontology creation process is less volatile and more consistent. The following improvements suggest ways to accomplish these goals.

The ontology should maintain all of the metric information from the other agents in the environment. This will ensure that any ontology updating that takes place, matches with all of the agents, not just the leader.

If agents could recognize that two fields in one database represent one combined field in another, the ontology created would be more accurate and more automated. There is no need for a human expert to go in and correct this relationship. The ontology would look something like, 'time' → 'day_time' and 'day' → 'day_time'.

SemInt, discussed in Section 2.7.4, uses numerical methods that operate on more metadata features than was done in this research. Combining the substring match with the information used by SemInt should produce better results than those found in Chapter 4.

Instead of leaving the ontology negotiation up to one agent, allow agents to cooperatively create the ontology eliminating the sole leader. Although this will result in more communication and memory requirements, this technique could also produce better results than those in Chapter 4.

In addition, exploring different distance similarity metrics could produce better results that are more consistent. The Jaro metric provided a way to see how feasible automated agent ontology creation was for string-based fields. There are other distance metrics, which could prove to be better and more accurate than the Jaro similarity metric method for numeric or number related fields.

Appendix A

Agent Attribute Values

Agent1	Agent2	Agent3
ACFT_QTY	ACFT_Quantity	Aircraft_QTY
ACFT_TYPE	Aircraft_TYPE	ACFT_TYPE
DATETIME_CREATED	DATETIME_CREATED	DATETIME_CREATED
DATETIME_LAST_CHG	DATETIME_LAST_CHG	DATETIME_LAST_CHG
PROB_DAMAGE_TOTAL	DAMAGE	PROB_DAMAGE
WPN_NAME	Weapon_NAME	WeaPoN_NAME
WPN_QTY	WPN_QTY	WPN_Quantity

Agent4	Agent5
CATEGORY	CAT
CONDITION	CC
COORD_DATUM	COLLATERAL_DAMAGE
COORD_DERIV	CONDITION
COORD_LT	COORD_LAT
COORD_LAT_ORD	COORD_LAT_ORD
COORD_LONG	COORD_L
COORD_L_ORD	COORD_LONG_ORD
COORD_ROA	DATE_CREATED
DATETIME_CREATED	TIME_LAST_CHANGED
DATETIME_LAST_CHG	FAC_NAME
DMPI_ID	FUNCT_PRIMARY
EVAL	NO_STRIKE
MSN_TYPE	OPER_STATUS
OPER_STATUS	REMARK
TGT_DTL_NAME	
TGT_OBJ_NAME	

Appendix B

Oracle Data Agent Attribute Values

Agent1	**Agent2**	**Agent3**
MSN_WW_ID	Mission	MissionID
AIR_MSN_EVNT_ID	MISSION_EVENT	EVENT_ID
ABP_WW_ID	ABP_WW_ID	WW_ID
ABP_REQ_ID	REQ_ID	ABP_ID
AMO_ID	AMO_ID	AMO_ID
AIR_MSN_EVNT_ACTUAL_DTTM	ACTUAL_DTTM	DATETIME

Appendix C

String Only Agent Attribute Values

Agent1	Agent2	Agent3
CC	BB	AA
COLLATERAL_DAMAGE	XTRA_DAMAGE	OTHER_DMG
CONDITION	COND	STATE
FAC_NAME	FACILITY	NAME_OF_FACILITY
FUNCT_PRIMARY	PRIMARY_FUNCTION	TARGET_TYPE
OPER_STATUS	O_STAT	OPERATOR
REMARK	COMMENTS	ADD_INFO

Bibliography

1. Kowalchuk, A., Implementing an Information Retrieval and Visualization Framework for Heterogeneous Data Types. Masters Thesis. Department of Computer Systems, Air Force Institute of Technology, Wright Patterson Air Force Base OH, 2003.

2. Roell, R., A Data Framework for Integrating Heterogeneous Systems Using Agents, XML, and CoABS. Masters Thesis. Department of Computer Systems, Air Force Institute of Technology, Wright Patterson Air Force Base OH, 2003.

3. United States Air Force Scientific Advisory Board. Information Management to Support the Warrior. Report SAB-TR-98-02, December 1998.

4. Department of Defense. Joint Vision 2020. Chairman Joint Chief of Staff. 20 September 2001.

5. Hendler, J., Agents and the Semantic Web. University of Maryland, 2001.

6. Marmelstein, R., Force Templates: A Blueprint for Coalition Interaction within an Infosphere. Air Force Research Laboratory, Rome, NY, May/June 2002.

7. Milligan, J., and J. Hendler. JBI Fuselet Definition Document. Air Force Research Laboratory, Rome, NY, Draft – May 2003.

8. Kindler, C., Jini-Based Publish and Subscribe for JBI Clients. ITT Industries, Advanced Engineering and Sciences, 2002.

9. Kavi, K., , M. Aborizka, , and D. Kung, A Framework For Designing, Modeling and Analyzing Agent Based Software Systems. 2002.

10. Sun Microsystems. Jini Network Technology. 2001 Palo Alto, CA, http://wwws.sun.com/software/jini/whitepapers/jini-datasheet0601.pdf.

11. Kahn, M., and C. Cicalese. DARPA CoABS Grid Users Manual. http://coabs.globalinfotek.com/public/downloads/Grid/documents/GridUsersManual.v4- draft.doc. October 2002.

12. Gruber, T., A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220, 1993.

13. DiLeo, J., Ontological Engineering and Mapping in Multiagent Systems Development. Masters Thesis. Department of Computer systems, Air Force Institute of Technology, Wright Patterson Air Force Base OH, 2002.

14. Han, J., and M. Kamber, Data Mining Concepts and Techniques. Academic Press, San Diego, CA, 2001.

15. Cohen, W., P. Ravikumar, and S. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. Center for Automated Learning and Discovery, School of Computer Science, Carnegie Mellon University, 2003.

16. Machado, C., and K. Hill, Probabilistic Record Linkage and an Automated Procedure to Minimize the Undecided-Matched Pair Problem. The Bloomberg School of Public Health, Johns Hopkins University, 2003.

17. Charras, C. and T. Lecroq, Boyer-Moore Algorithm. http://www-igm.univ-mlv.fr/~lecroq/string/node14.html, 1997.

18. Miller, G., C. Fellbaum, R. Tengi, S. Wolff, P. Wakefield, H. Langone, and B. Haskell, WordNet. Cognitive Science Laboratory, Princeton University, http://www.cogsci.princeton.edu/~wn/index.shtml, 2003.

19. Weiss, G., Multi-Agent Systems. MIT Press, Cambridge MA, 1999.

20. Arai, S., K. Sycara, and T. Payne, Multi-Agent Reinforcement Learning for Planning and Scheduling Multiple Goals. 1-2, 1999.

21. Sycara, K. and A. Ankolekar. Retsina. The Intelligent Software Agents Lab – The Robotics Institute, Carnegie Mellon University, http://www-2.cs.cmu.edu/~softagents/retsina.html, 2001.

22. Clifton, C., E. Housman, and A. Rosenthal, Experience with a Combined Approach to Attribute-Matching Across Heterogeneous Databases. Chapman and Hall Press, 1997.

23. Berlin, J. and A. Motro, Database Schema Matching Using Machine Learning with Feature Selection. Information and Software Engineering Department, George Mason University, 2002.

24. Do, H. and E. Rahm, COMA – A System for Flexible Combination of Schema Matching Approaches. University of Leipzig, 2002.

25. Madhavan, J., P. Bernstein, and E. Rahm, Generic Schema Matching with Cupid. Proceedings of the 27th VLDB Conference, Roma, Italy, 2001.

26. Madhavan, J., P. Bernstein, K. Chen, A. Halevy, and P. Shenoy. Corpus-based Schema Matching. University of Washington, 2003.

27. Bouquet, P., L. Serafini, and S. Zanobini. Semantic Coordination: A New Approach and an Application, Department of Information and Communication Technology, University of Trento, 2003.

28. Williams, A., and C. Tsatsoulis. An Instance-based Approach for Identifying Candidate Ontology Relations within a Multi-Agent System. Department of Electrical and Computer Engineering, University of Iowa, 2001.

29. Doan, A., P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2001.

30. Do, H., S. Melnik, and E. Rahm, Comparison of Schema Matching Evaluations. University of Leipzig, Germany, 2003.

31. Yatskevich, M., Preliminary Evaluation of Schema Matching Systems. Department of Information and Communication Technology, University of Trento, Italy, 2003.

Vita

First Lieutenant Austin A. Bartolo graduated from Aquinas High School in Southgate, Michigan. He entered undergraduate studies at Chapman University in Tucson, Arizona where he graduated with a Bachelor of Science degree in Computer Science in May 1997. He was commissioned through OTS at Maxwell AFB, Alabama.

Before earning his commission, Austin spent 13 years enlisted in the Air Force. His first assignment was at Malmstrom AFB, Montana where he worked as an air cargo specialist in March 1987. In November 1989, he was assigned to the 616th Aerial Port Squadron, Galena APT, Alaska where he served as an air cargo specialist. From 1990 to 1995, Austin Bartolo worked in the Traffic Management Office at Davis-Monthan AFB, Arizona. In February 1995 he became a member of the 630th Air Mobility Support Squadron at Yokota AB, Japan and spent four years as a passenger service representative. In February 1999 he was stationed at the 60th Aerial Port Squadron at Travis AFB, California where he served as the squadron network NCOIC. In April 2000, Austin served as the officer in charge of the Information Systems Office in the Network Control Center at Scott AFB, Illinois. In August 2002, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the Standard Systems Group, Maxwell Gunter Annex, Alabama.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)* 23-03-2004	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* March 2003 – March 2004

4. TITLE AND SUBTITLE AUTOMATED AGENT ONTOLOGY CREATION FOR DISTRIBUTED DATABASES	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Bartolo Austin A., First Lieutenant, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENG/04-01
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Labs / Information Directorate/IFTC Attn: Mr. Douglas Holzhauer / Air Force Material Command(AFMC) 26 Electronics Parkway / DSN: 587-4920 Rome, NY 13441-4514 / e-mail: Douglas.holzhauer@rl.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In distributed database environments, the combination of resources from multiple sources requiring different interfaces is a universal problem. The current solution requires an expert to generate an ontology, or mapping, which contains all interconnections between the various fields in the databases. This research proposes the application of software agents in automating the ontology creation for distributed database environments with minimal communication. The automatic creation of a domain ontology alleviates the need for experts to manually map one database to other databases in the environment. Using several combined comparison methods, these agents communicate and negotiate similarities between information sources and retain these similarities for client agent queries without the manual mapping of different data sources achieving an average accuracy of 57% before leader negotiation and an average accuracy of 61% after leader negotiation. The best matching accuracy achieved in a single test is 79%.

This is directly applicable to the Department of Defense (DOD) that possesses many systems which share information that enables the military to achieve their objectives. The DOD created an environment called the Joint Battlespace Infosphere (JBI) to solve this integration problem. This research improves upon the JBI's use of exact matching of field names for integrating the information within the environment. It simulates this type of interaction by demonstrating agents wrapped around different databases negotiating and generating an ontology. An agent-generated ontology is compared with an expert generated ontology and testing uses a set of queries run against the ontologies show that this technique can be useful in a distributed information environment.

15. SUBJECT TERMS

Distributed Databases, Agents, Ontology

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Peterson, Gilbert L., PhD
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	106	(937) 255-6565, ext 4281 (emailname@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18